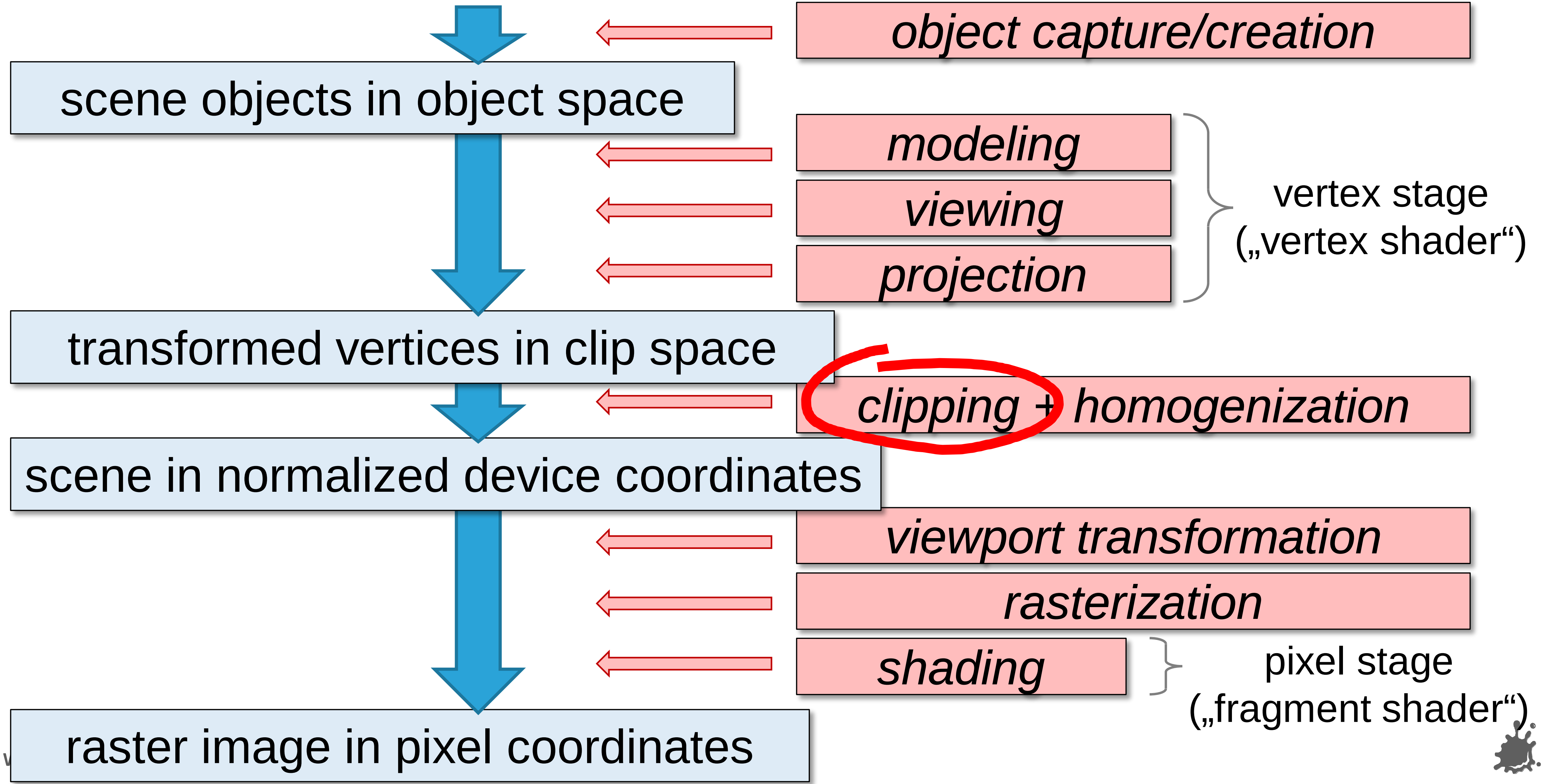
Einführung in Visual Computing

186.822

Clipping

Werner Purgathofer

Viewing in the Rendering Pipeline



scene objects in object space

transformed vertices in clip space

scene in normalized device coordinates

raster image in pixel coordinates

object capture/creation

modeling

viewing

projection

vertex stage („vertex shader“)

clipping + homogenization

viewport transformation

rasterization

shading

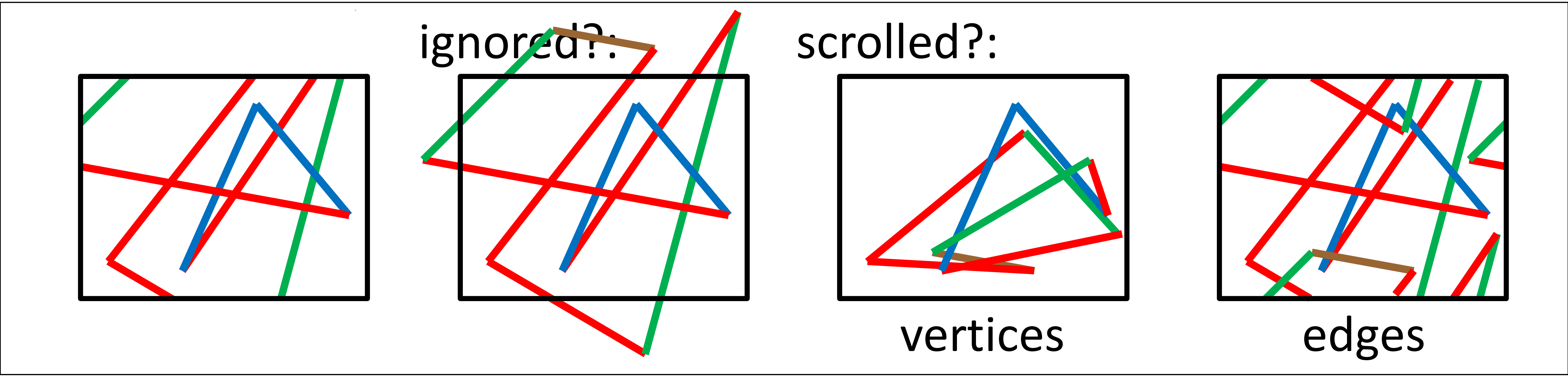
pixel stage („fragment shader“)

Overview: Clipping

- line clipping
- polygon clipping
- triangle clipping

Clipping

- partly visible or completely invisible parts
- must not be ignored and must not be drawn



- ⇒ must be cut off (as early as possible)

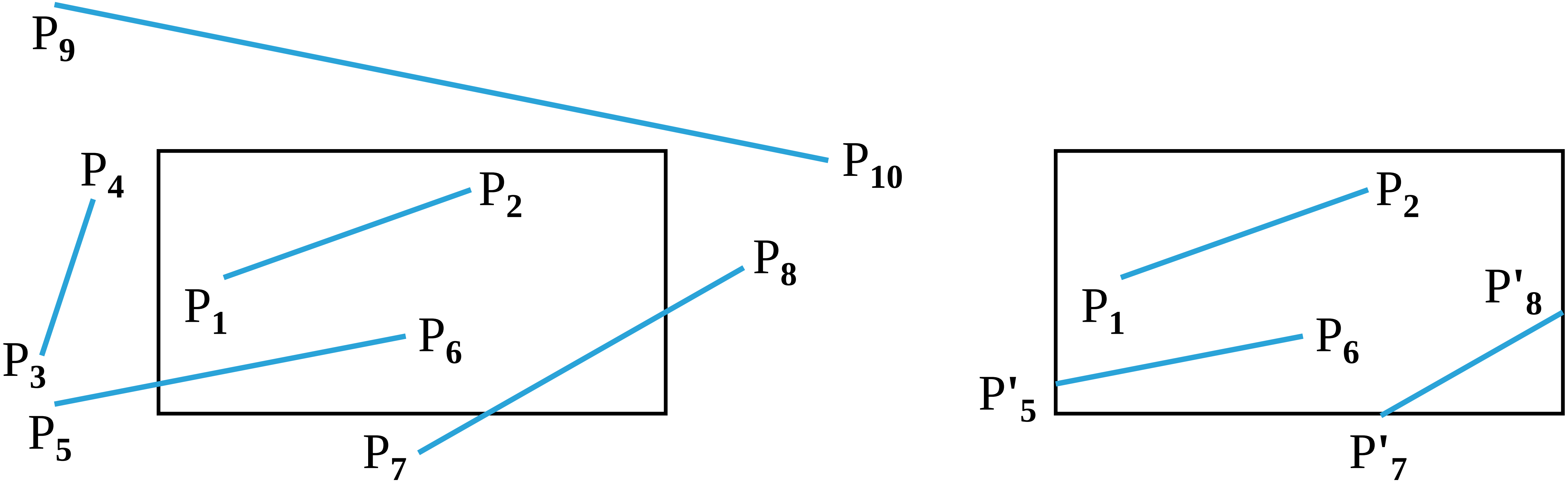
Clipping Operations

- remove objects outside a clip window
  - clip window: rectangle, polygon, curved boundaries
  - applied somewhere in the viewing pipeline
  - can be combined with scan conversion
  - objects to clip: points, lines, triangles, polygons, curves, text, ...

3 Principle Possibilities for Clipping

- *analytically* in *world* coordinates
  - reduces WC → DC transformations
- *analytically* in *clip* coordinates
  - simple comparisons
- *during raster conversation*
  = as part of the rasterization algorithm
  - may be efficient for complex primitives

Line Clipping (1)



before clipping

after clipping

[line clipping against a rectangular clip window]

Line Clipping (2)

- goals
  - eliminate simple cases fast
  - avoid intersection calculations

for endpoints (x_0,y_0), (x_{end},y_{end})

intersect parametric representation

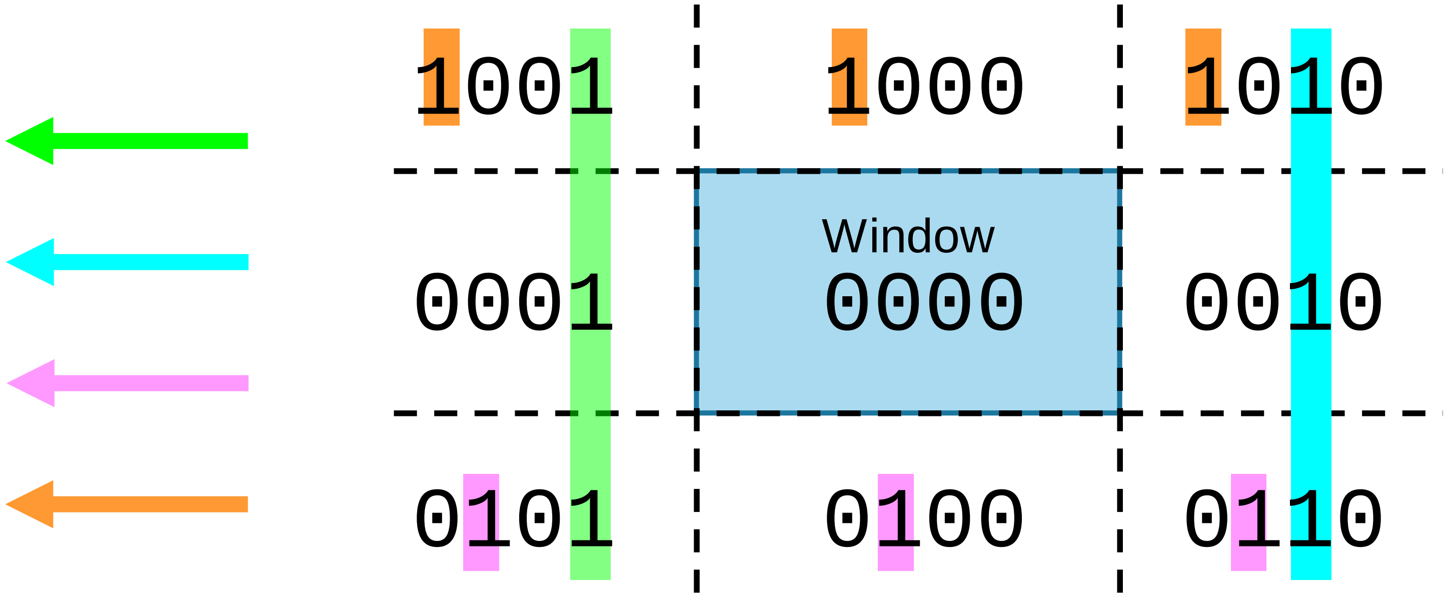
$$\begin{pmatrix}x\\y\end{pmatrix} = \begin{pmatrix}x_0\\y_0\end{pmatrix} + u\cdot\begin{pmatrix}x_{end} - x_0\\y_{end} - y_0\end{pmatrix}$$

with window borders:

intersection ⇔ $0 < u < 1$

Cohen-Sutherland Line Clipping

- assignment of region codes to line vertices
  - bit1: left
  - bit2: right
  - bit3: below
  - bit4: above



1001	1000	1010
0001	Window 0000	0010
0101	0100	0110

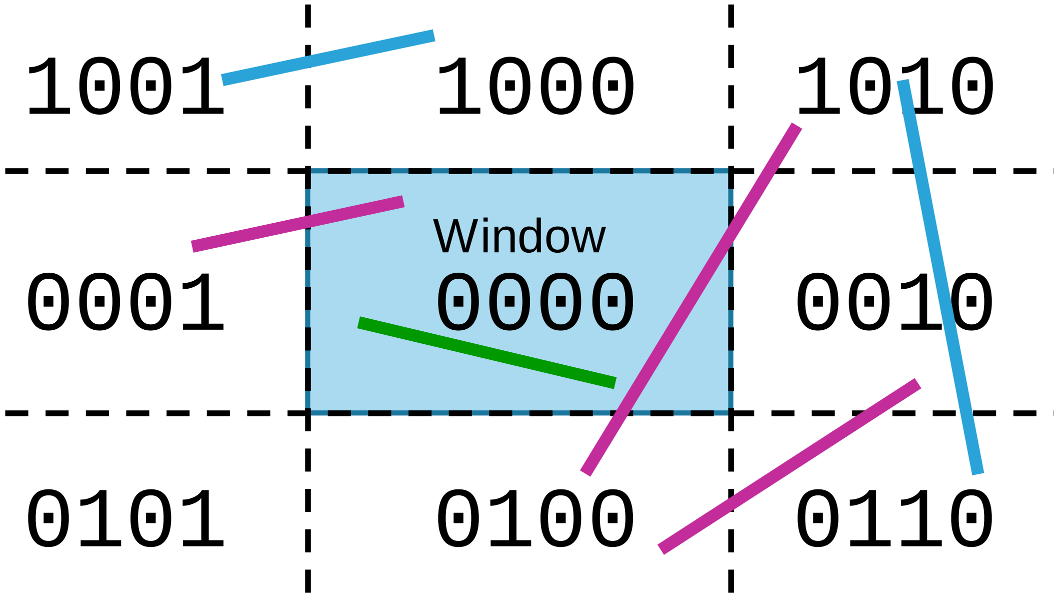
binary region codes assigned to line endpoints according to relative position with respect to the clipping rectangle

Cohen-Sutherland Line Clipping

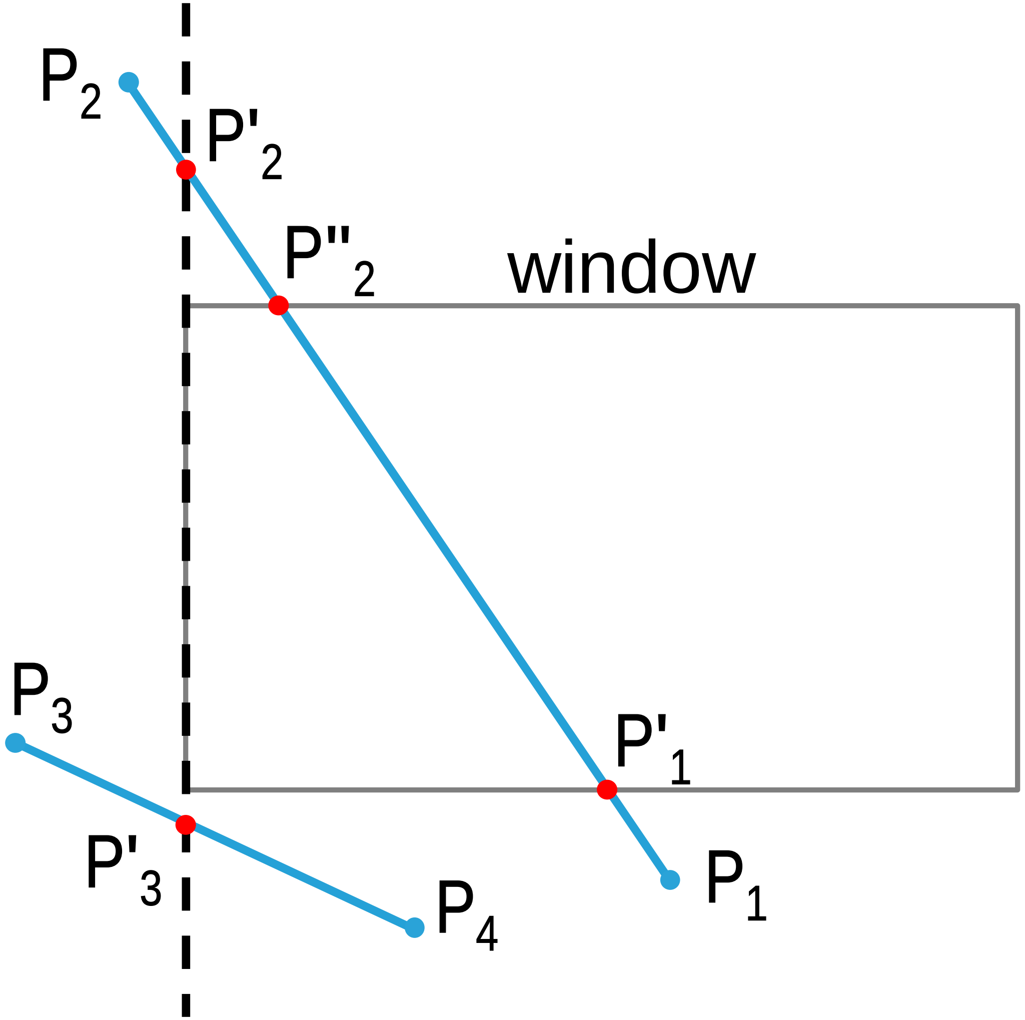
- “or” of codes of both points
  = 0000 ⇒ line entirely ***visible***

- “and” of codes of both points
  ≠ 0000 ⇒ line entirely ***invisible***

- all others ⇒ ***intersect!***



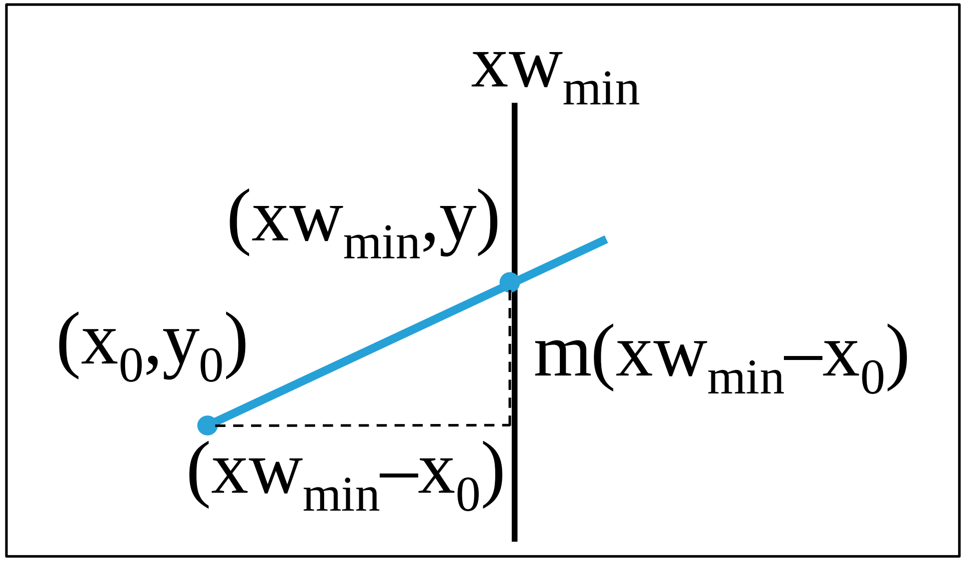
Cohen-Sutherland Line Clipping



lines extending from one coordinate region to another may pass through the clip window, or they may intersect clipping boundaries without entering the window

Cohen-Sutherland Line Clipping

- remaining lines
  - intersection test with bounding lines of clipping window
  - left, right, bottom, top
  - discard an outside part
  - repeat intersection test up to four times



vertical: $(x = xw_{min})$

$$y = y_0 + m(xw_{min} - x_0)$$

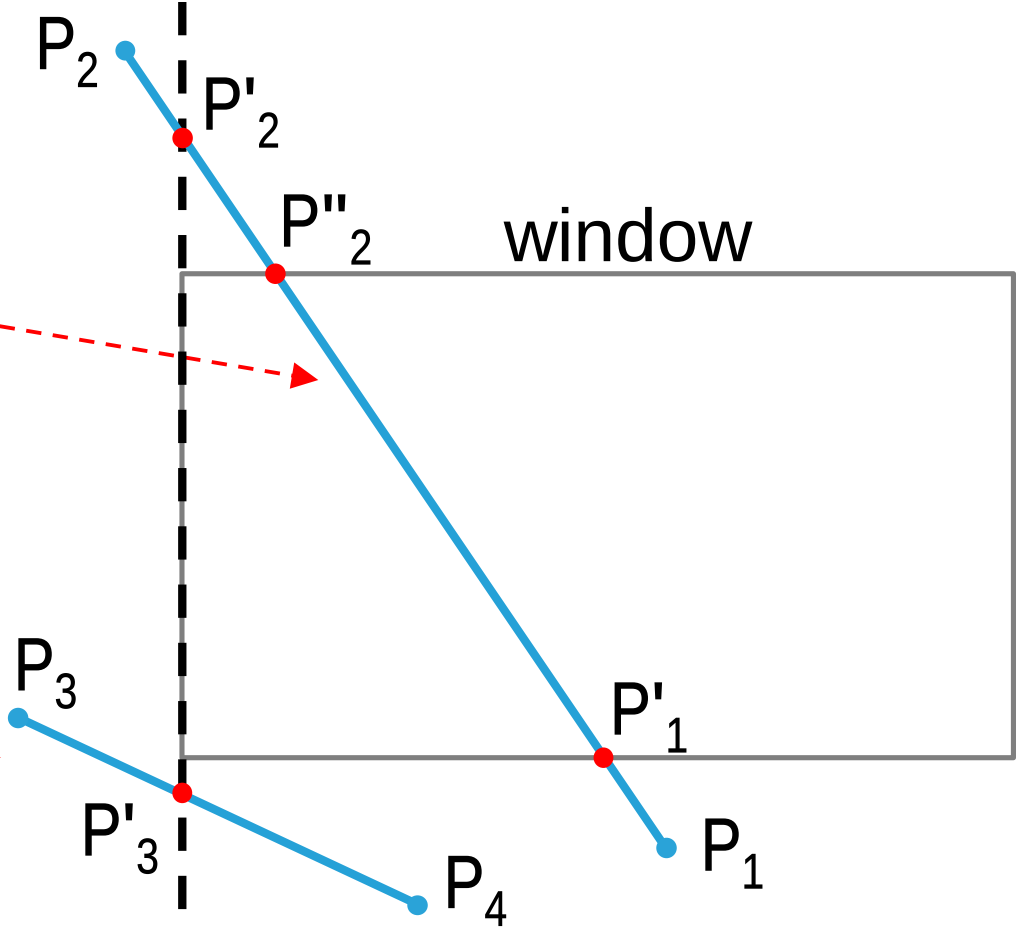
vertical: $y = y_0 + m(xw_{min} - x_0)$, $y = y_0 + m(xw_{max} - x_0)$

horizontal: $x = x_0 + (yw_{min} - y_0)/m$, $x = x_0 + (yw_{max} - y_0)/m$

Cohen-Sutherland Line Clipping

passes through clipping window

intersects boundaries without entering clipping window

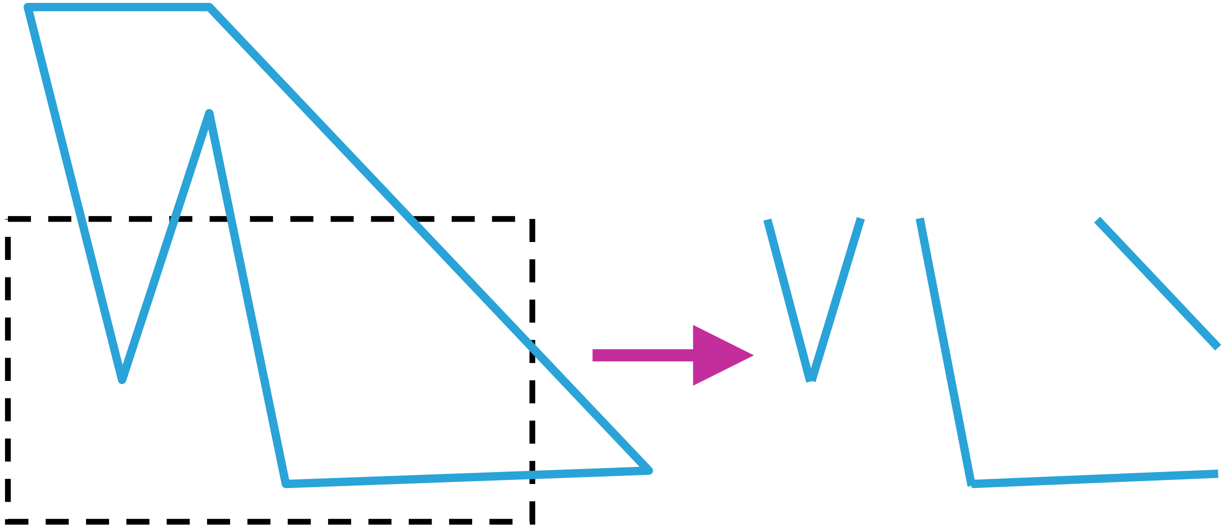


vertical: $y = y_0 + m(xw_{min} - x_0)$, $y = y_0 + m(xw_{max} - x_0)$

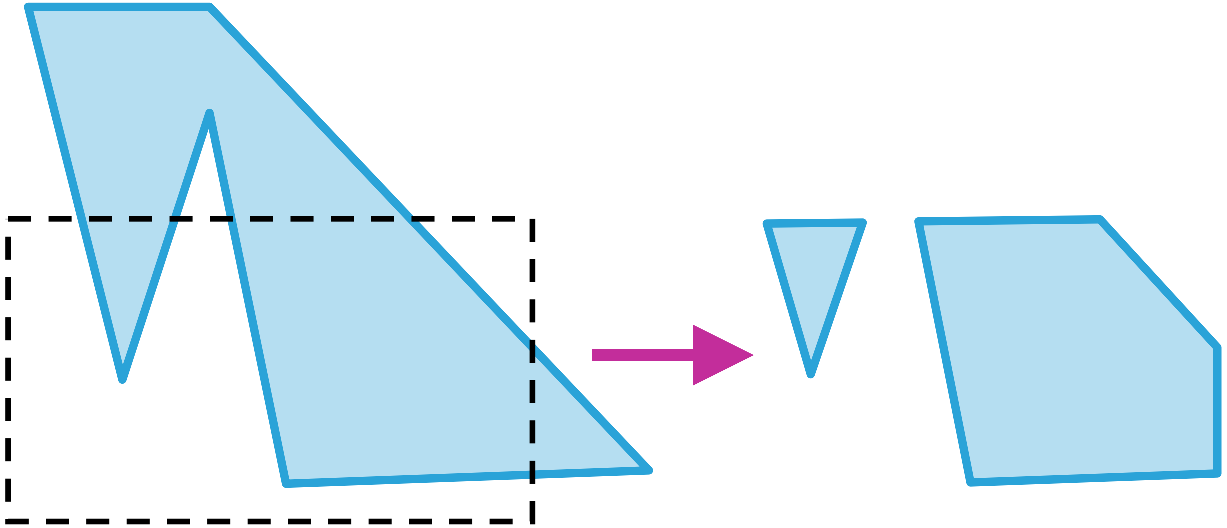
horizontal: $x = x_0 + (yw_{min} - y_0)/m$, $x = x_0 + (yw_{max} - y_0)/m$

Polygon Clipping

- modification of line clipping
- goal: one or more closed areas



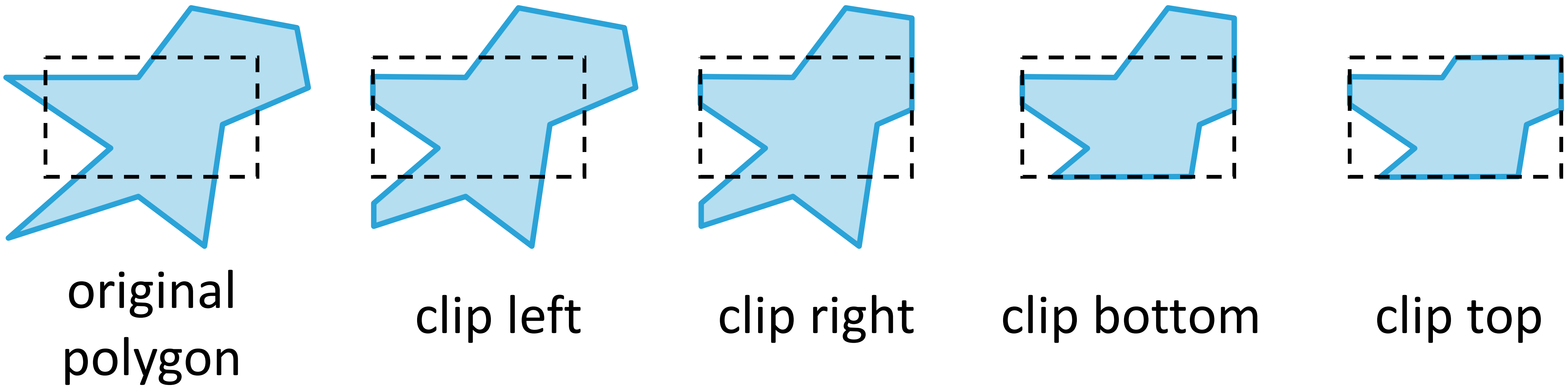
display of a polygon processed by a line-clipping algorithm



display of a correctly clipped polygon

Sutherland-Hodgman Polygon Clipping

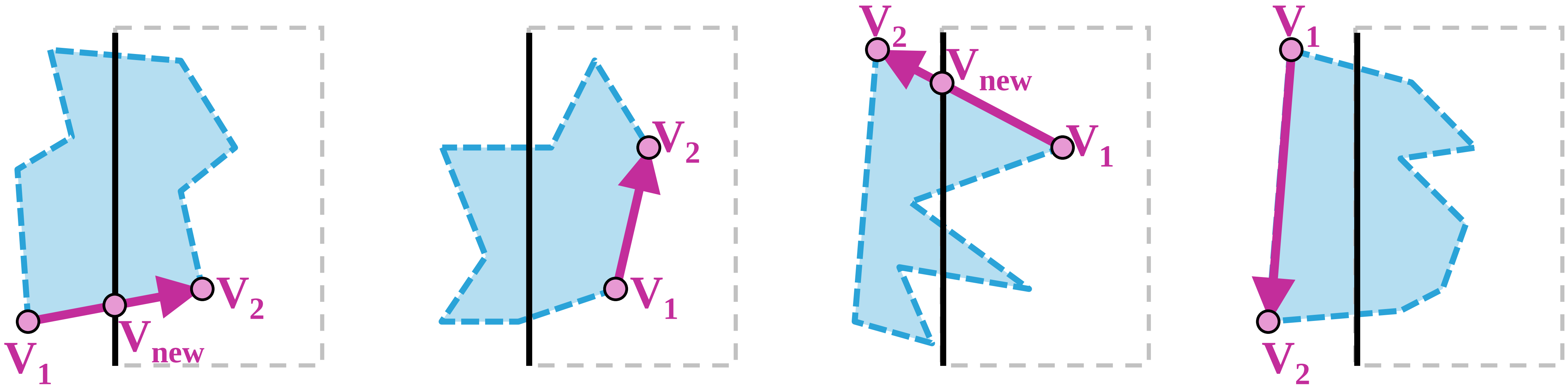
- processing polygon boundary as a whole against each window edge
- output: list of vertices



clipping a polygon against successive window boundaries

Sutherland-Hodgman Polygon Clipping

- four possible edge cases

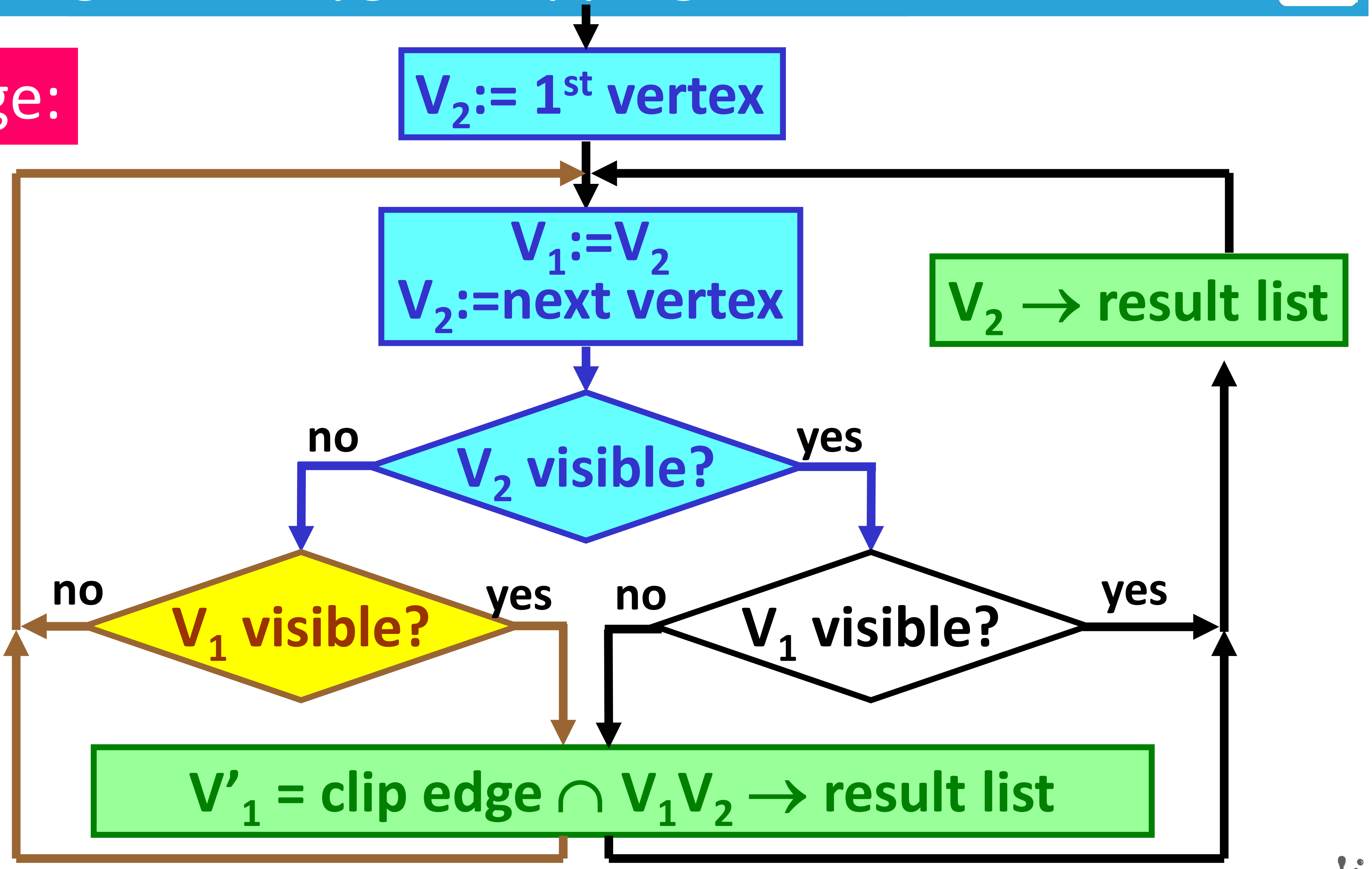


out → in	in → in	in → out	out → out
output: V_{new}, V_2	V_2	V_{new}	no output

successive processing of pairs of polygon vertices against the left window boundary

Sutherland-Hodgman Polygon Clipping

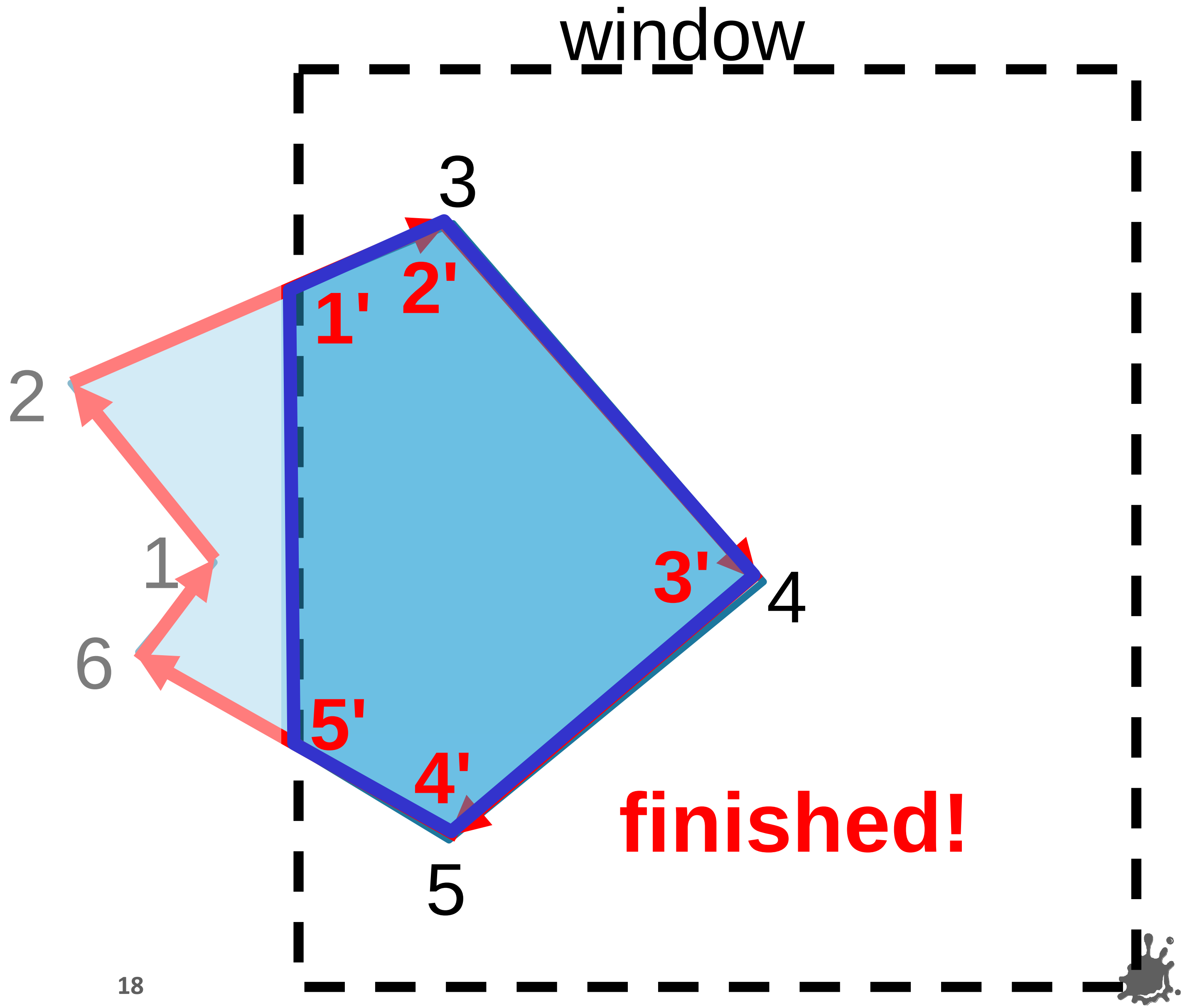
for 1 edge:



Sutherland-Hodgman Polygon Clipping

clipping a polygon against the left boundary of a window, starting with vertex 1.

primed numbers are used to label the points in the output vertex list for this window boundary



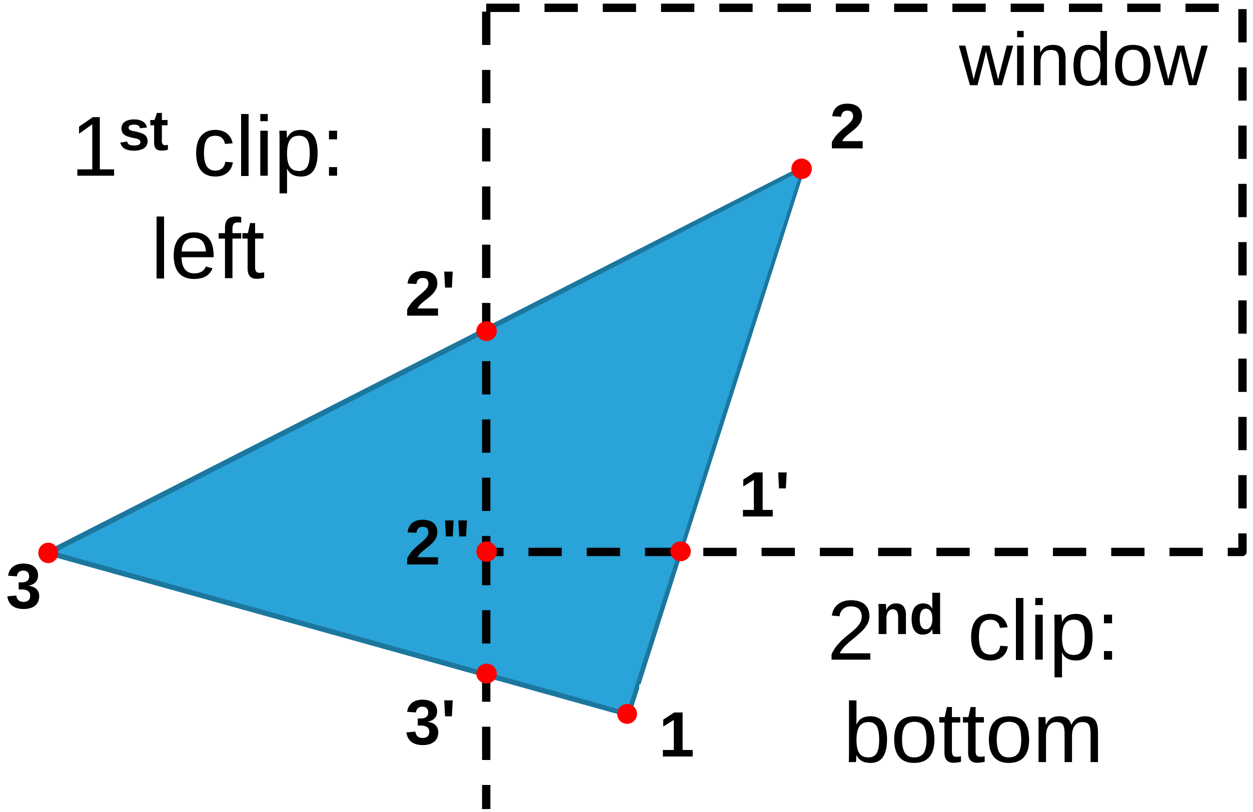
Polygon Clip: Combination of 4 Passes

- the polygon is clipped against each of the 4 borders separately, that would produce 3 intermediate results.

- by calling the 4 tests *recursively*,
  (or by using a clipping pipeline)
  every result point is immediately processed on,
  so that only *one* result list is produced

Sutherland-Hodgman Clipping Example

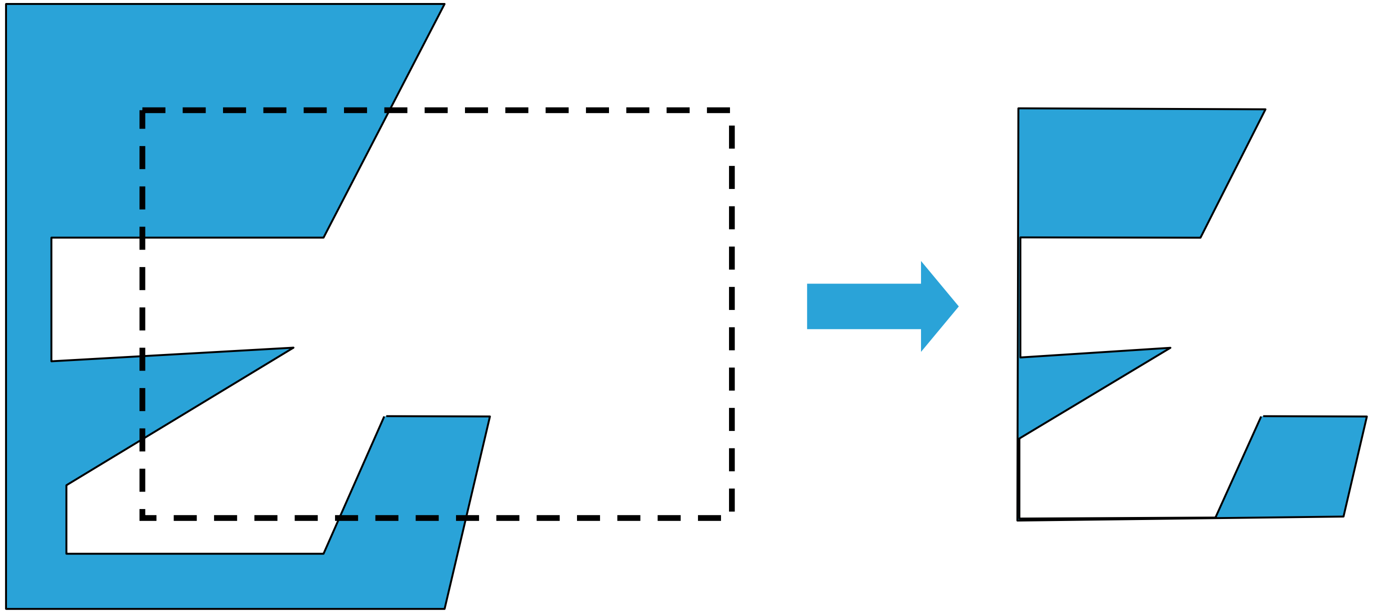
- pipeline of boundary clippers to avoid intermediate vertex lists



Processing the polygon vertices through a boundary-clipping pipeline. After all vertices are processed through the pipeline, the vertex list for the clipped polygon is {1', 2, 2', 2"}

Sutherland-Hodgman Polygon Clipping

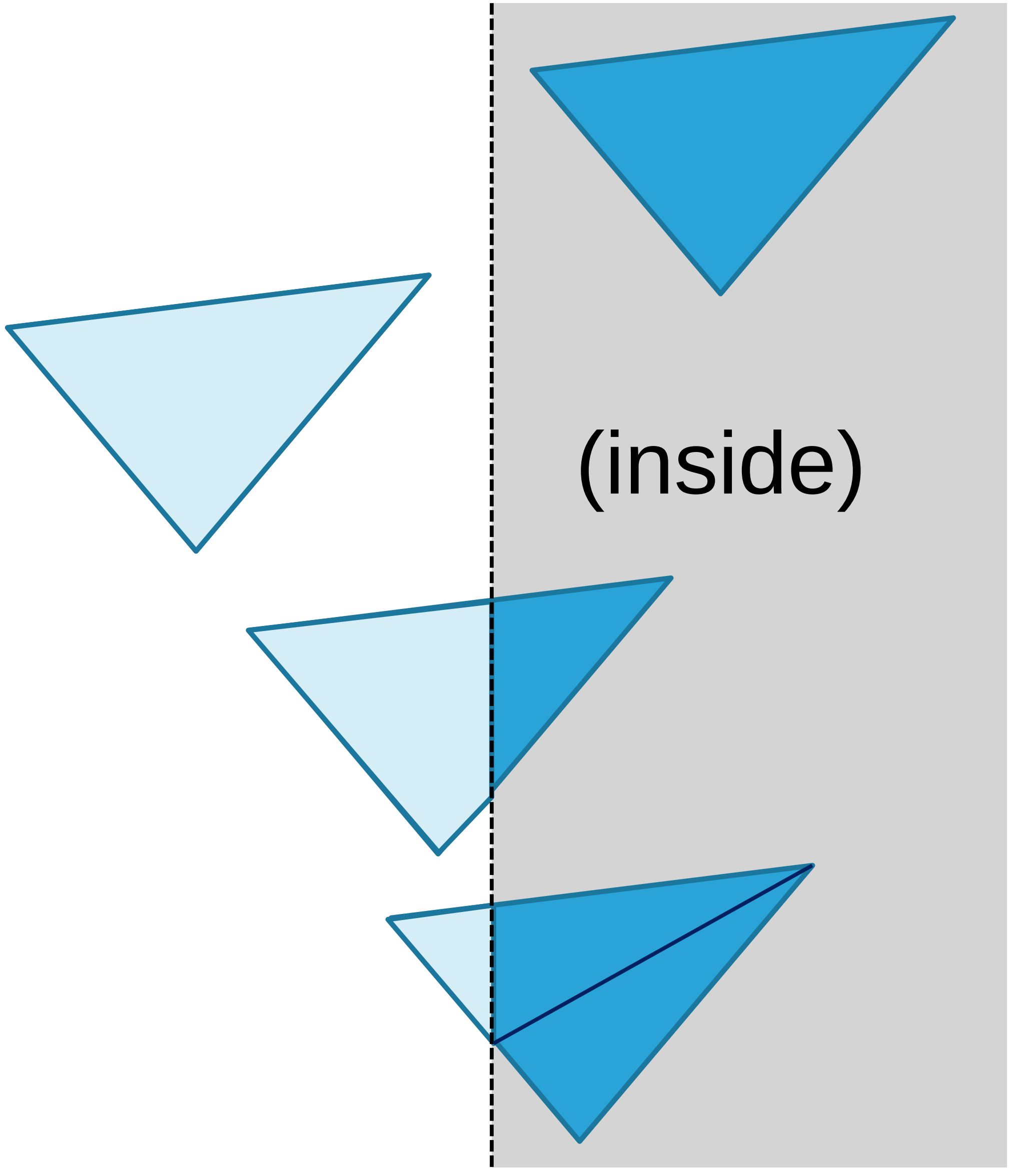
- extraneous lines for concave polygons:
  - split into separate parts or
  - final check of output vertex list



clipping the concave polygon with the Sutherland-Hodgeman clipper produces three connected areas

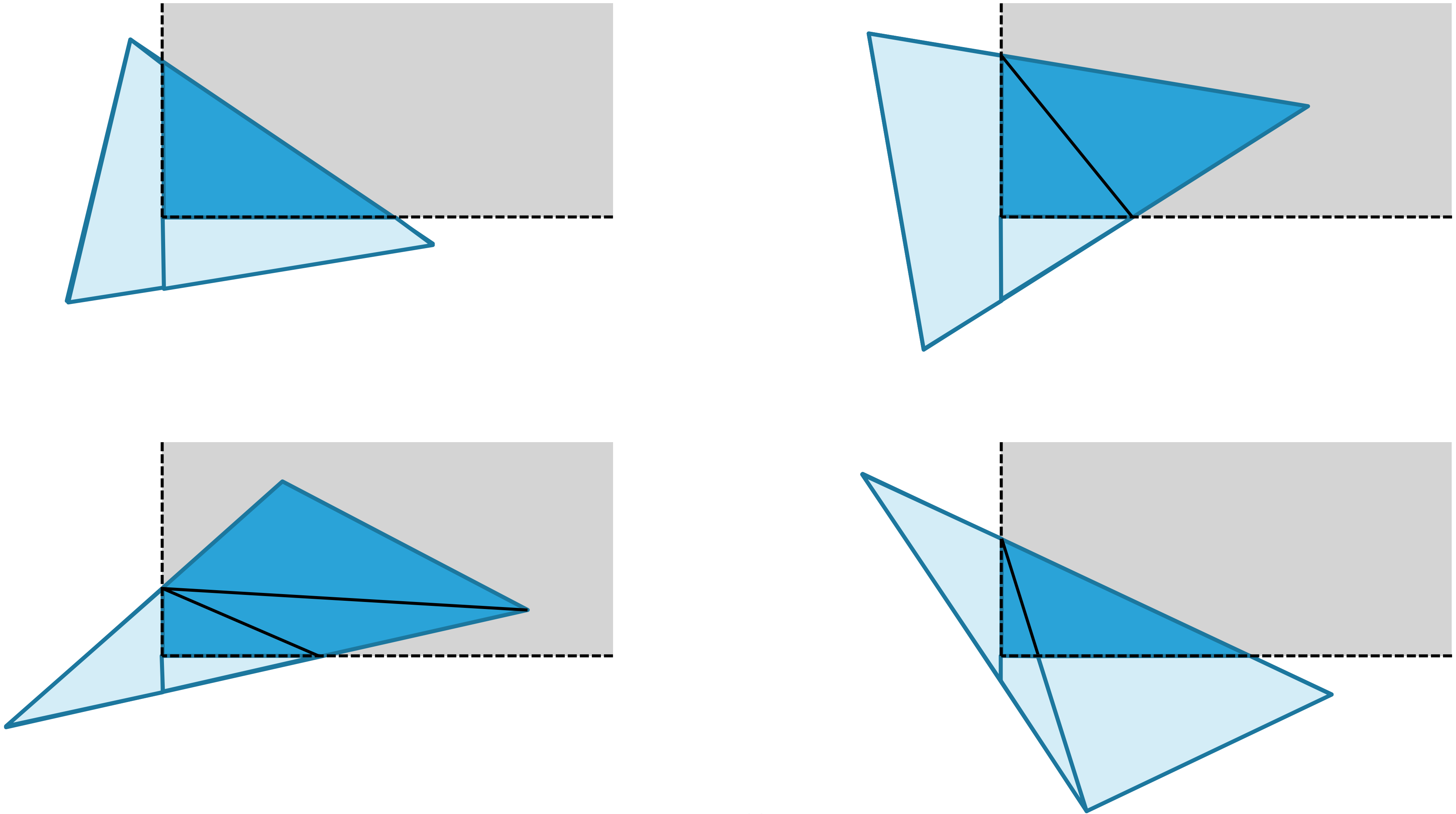
Clipping of Triangles

- often b-reps are “triangle soups”
- clipping a triangle → triangle(s)
- 4 possible cases:
  - inside
  - outside
  - triangle
  - quadrilateral → 2 triangles

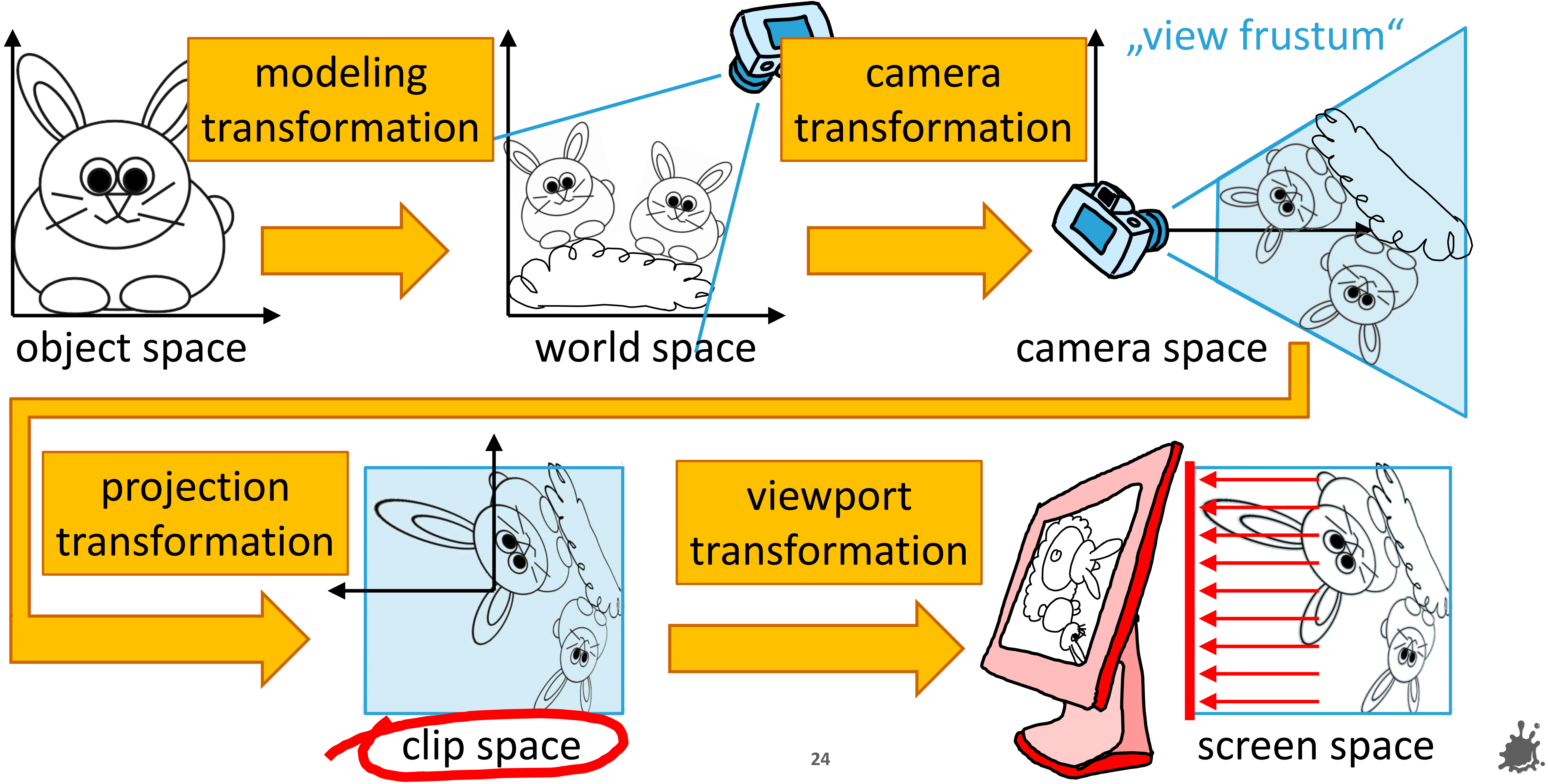


Clipping of Triangles

- corner cases need no extra handling!



From Object Space to Screen Space



modeling transformation

object space

world space

camera transformation

„view frustum“

camera space

projection transformation

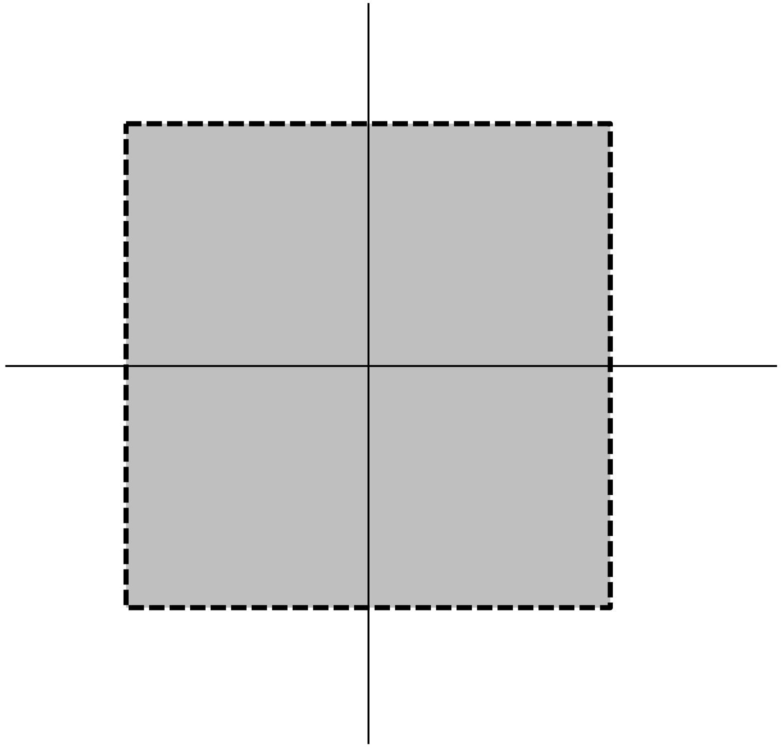
clip space

viewport transformation

screen space

Clipping in Clip-Space

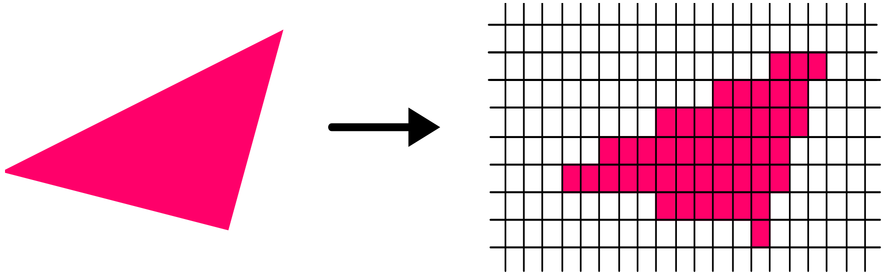
- clipping against $x = \pm 1, \quad y = \pm 1, \quad z = \pm 1$
- (x,y,z) inside?
- → only compare one value per border!

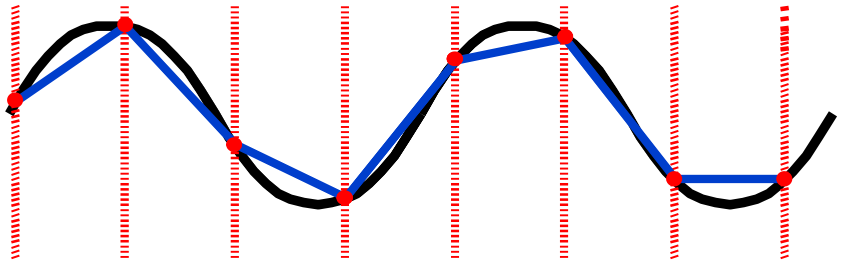


- is done *before* homogenization:
  - $x = \pm h, \quad y = \pm h, \quad z = \pm h$
  - clips points that are behind the camera!
  - reduces homogenization divisions

Einführung in Visual Computing

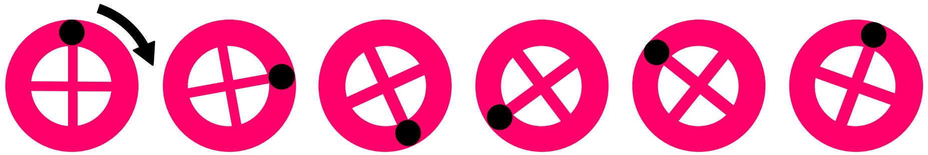
186.822



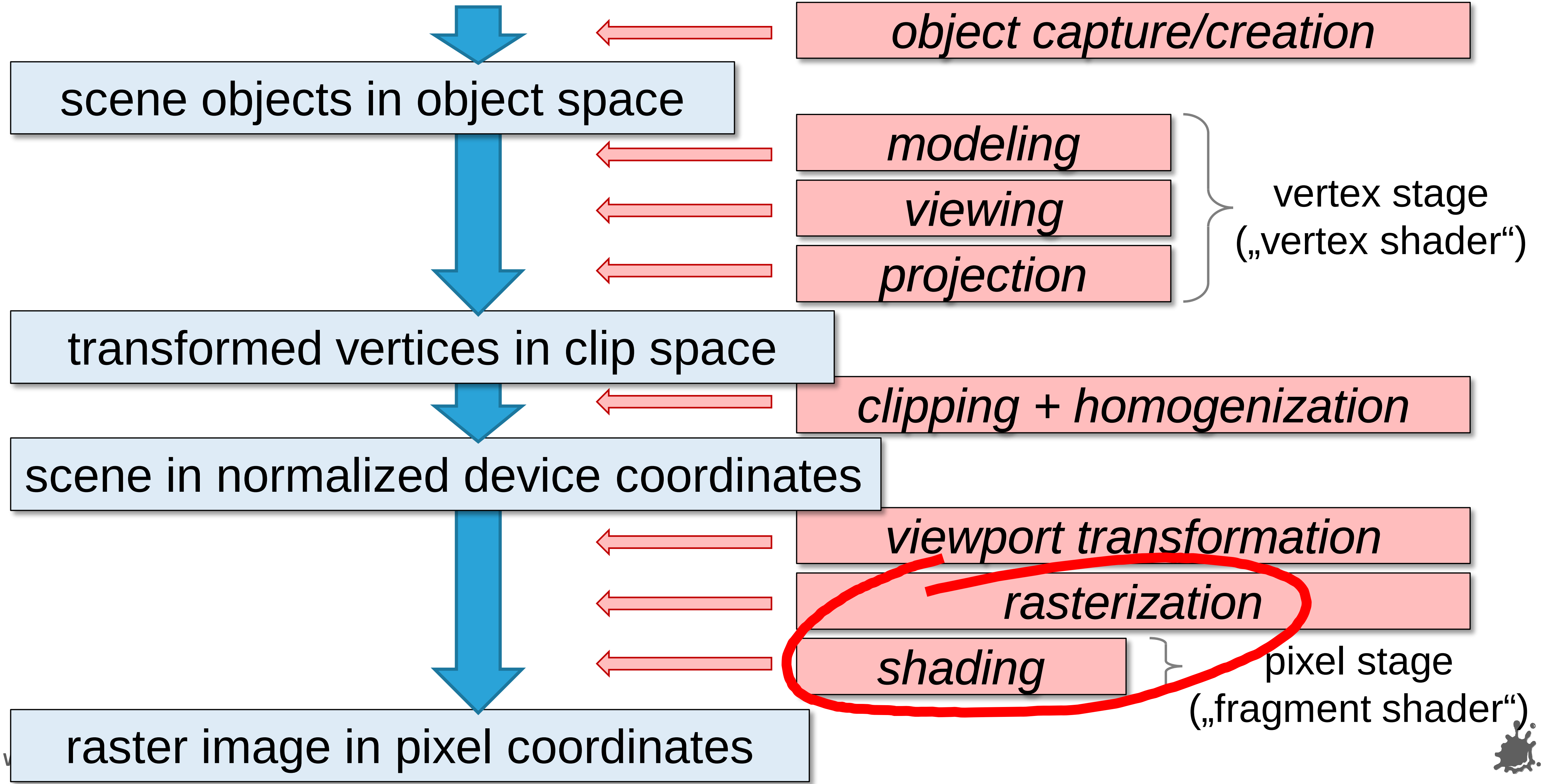


Antialiasing

Werner Purgathofer



Antialiasing in the Rendering Pipeline



scene objects in object space

transformed vertices in clip space

scene in normalized device coordinates

raster image in pixel coordinates

object capture/creation

modeling

viewing

projection

vertex stage („vertex shader“)

clipping + homogenization

viewport transformation

rasterization

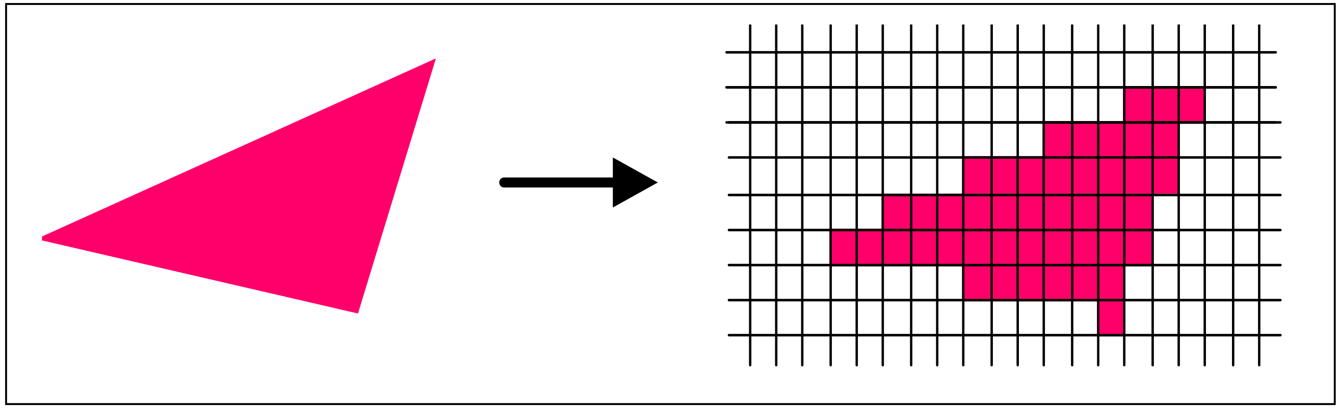
shading

pixel stage („fragment shader“)

Aliasing and Antialiasing

- what is aliasing? ['eiliæsiŋ]
- what is the reason for aliasing?
- what can we do against it?

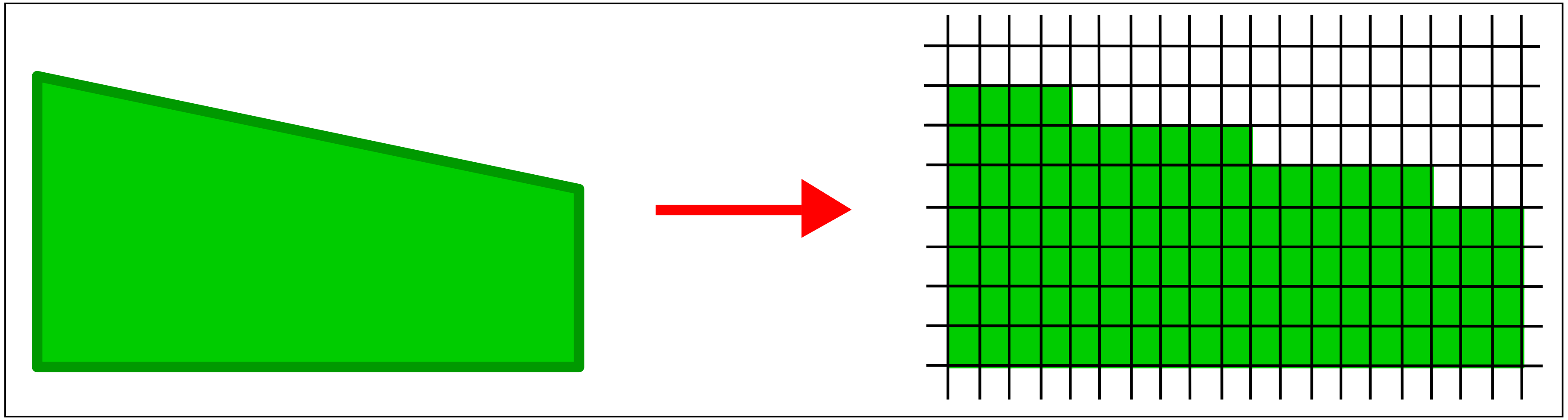
What is Aliasing?

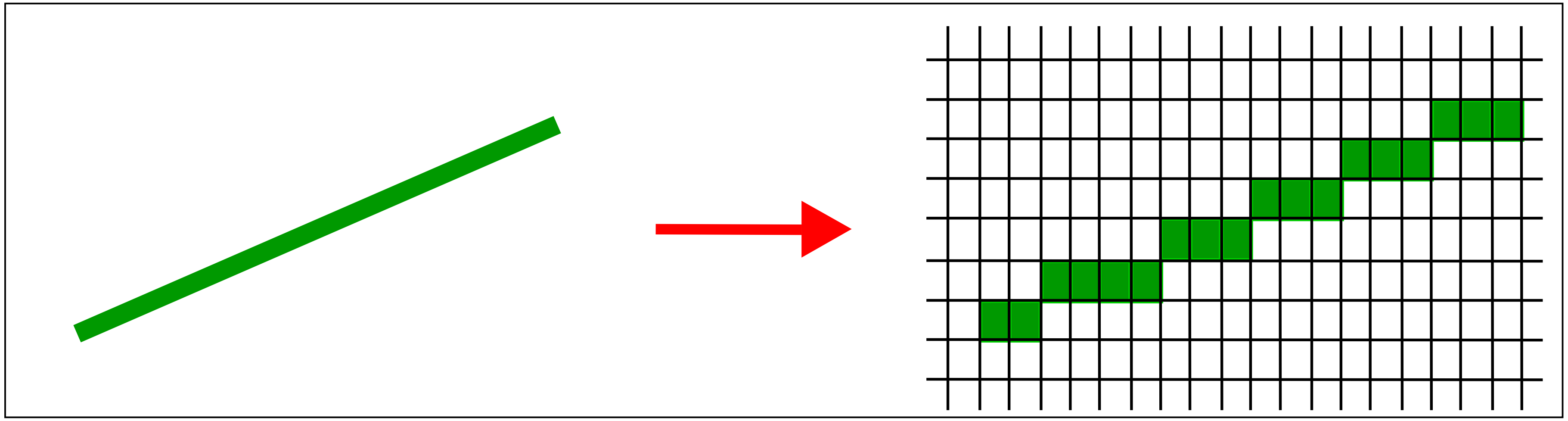


errors that are caused by the discretization of analog data to digital data

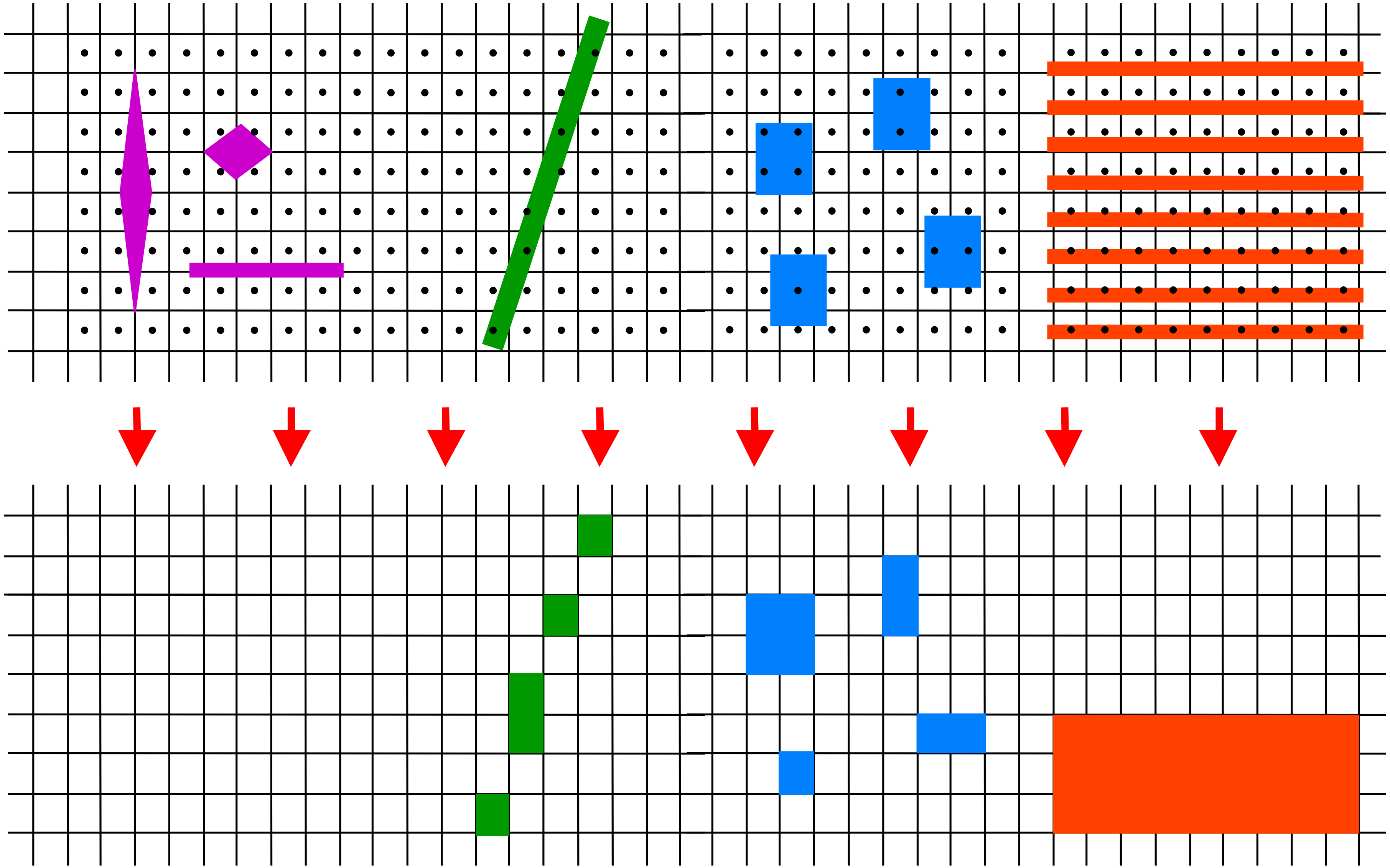
- *too bad resolution*
- *too few colors*
- *too few images / sec*
- *geometric errors*
- *numeric errors*

Aliasing: Staircase Effect



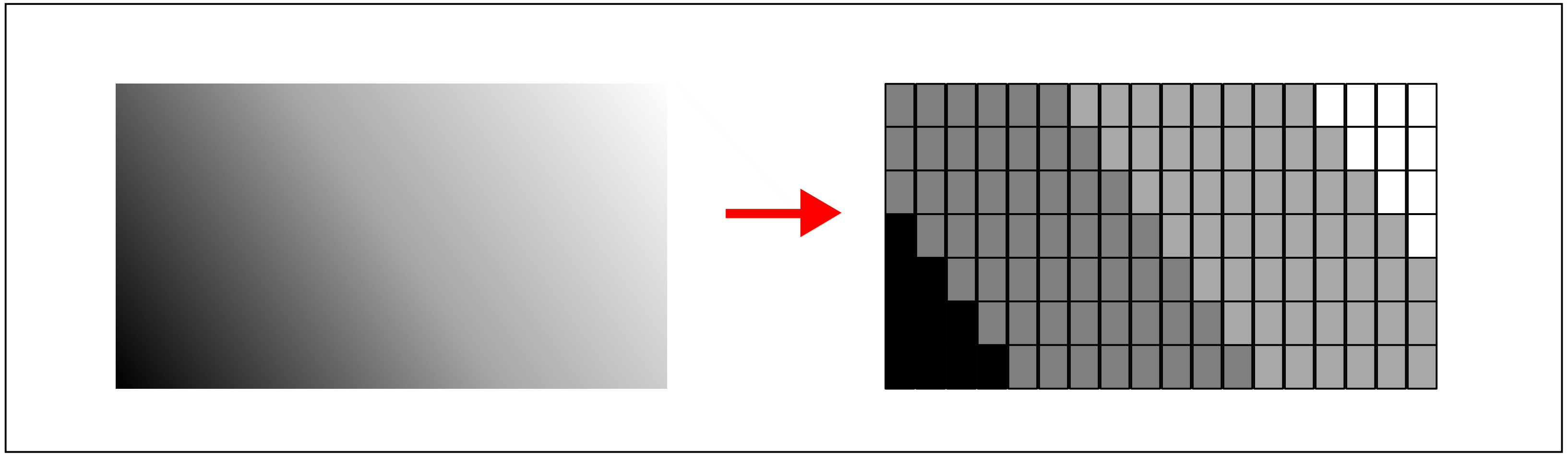


Various Aliasing Effects



Werner Purgathofer

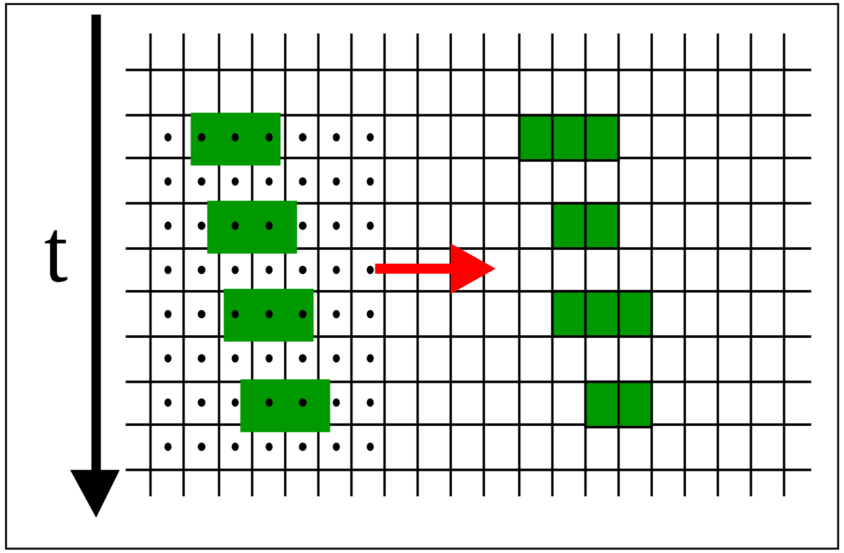
Aliasing from too few Colors

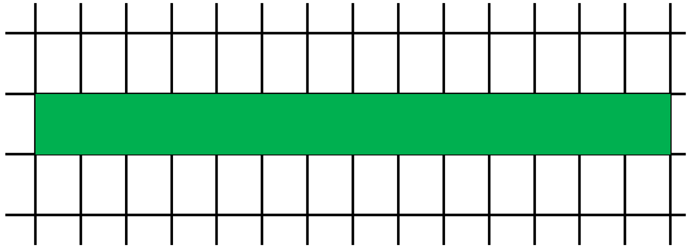


artificial color borders can appear

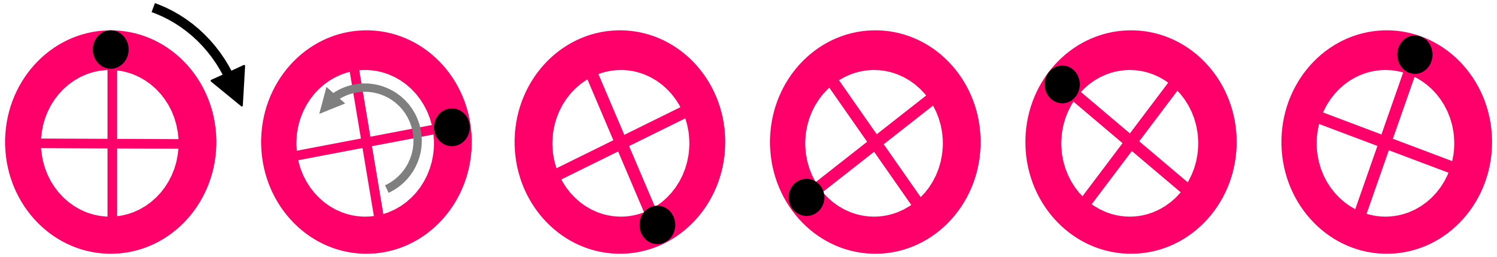
Aliasing in Animations

- jumping images
- "worming“





- backwards rotating wheels



Solutions against Aliasing?

- **1. improve the devices**
  - higher resolution
  - more color levels
  - faster image sequence

expensive
or
incompatible

- **2. improve the images = *antialiasing***
  - postprocessing
  - prefiltering !

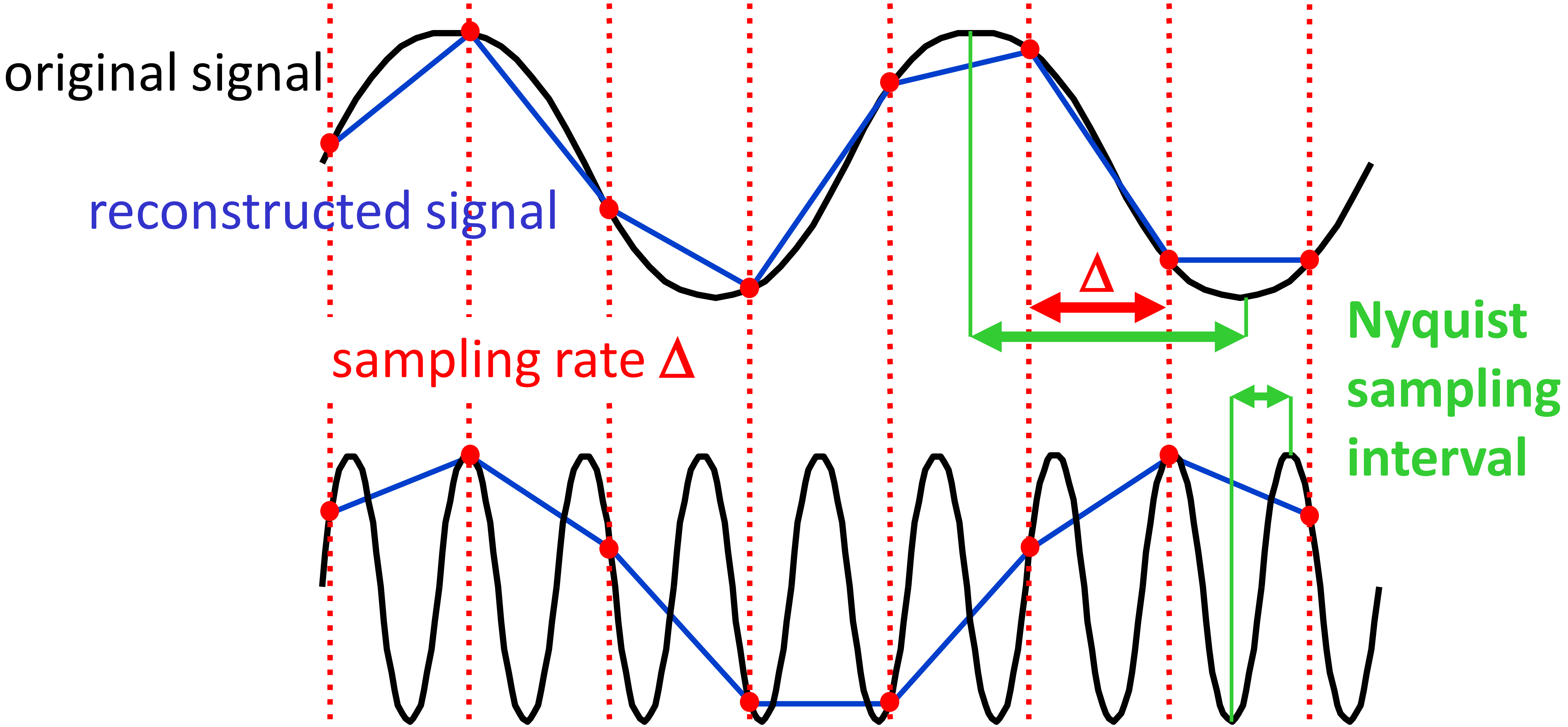
software !

Nyquist-Shannon Sampling Theorem

a signal can only be reconstructed without information loss
if the **sampling frequency** is at least
twice the highest frequency of the signal

this border frequency is called "**Nyquist Limit**"

Nyquist-Shannon Sampling Theorem



Antialiasing: Nyquist Sampling Frequency

a signal can only be reconstructed without information loss
if the **sampling frequency** is at least
twice the highest frequency of the signal

Nyquist sampling frequency: $f_s = 2\, f_{\max}$

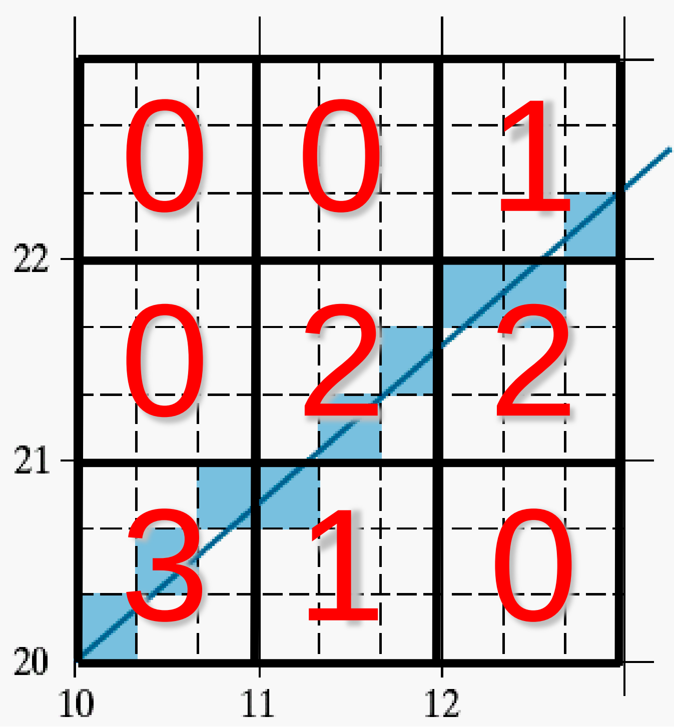
$$\Delta x_s = \frac{\Delta x_{cycle}}{2} \quad \text{with} \quad \Delta \mathrm{x}_{\mathrm{cycle}} = 1/\, f_{\max}$$

i.e. sampling interval ≤ one-half cycle interval

Antialiasing Strategies

- supersampling straight-line segments
- subpixel weighting masks
- area sampling straight-line segments
- filtering techniques
- compensating for line-intensity differences
- antialiasing area boundaries
  - (adjusting boundary pixel positions)
  - adjusting boundary pixel intensity

Antialiasing: Supersampling Lines

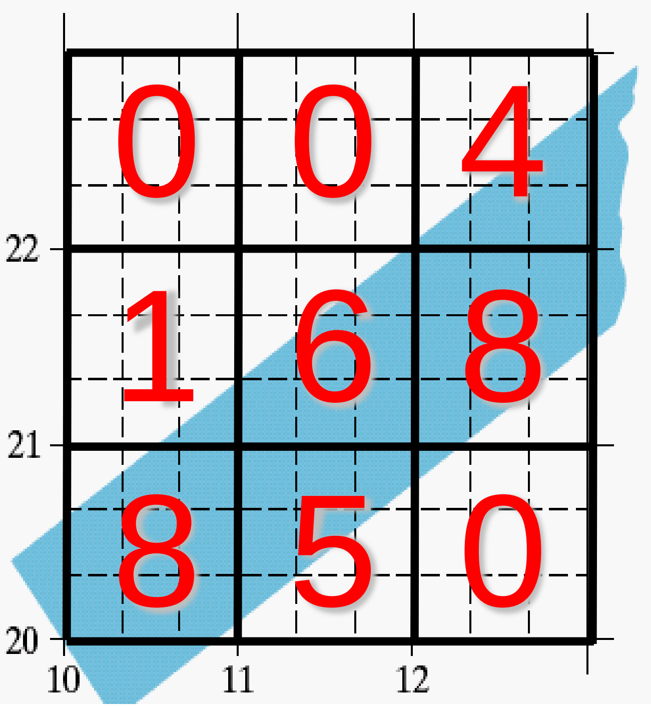


mathematical line

3 = max. intensity

...

0 = min. intensity



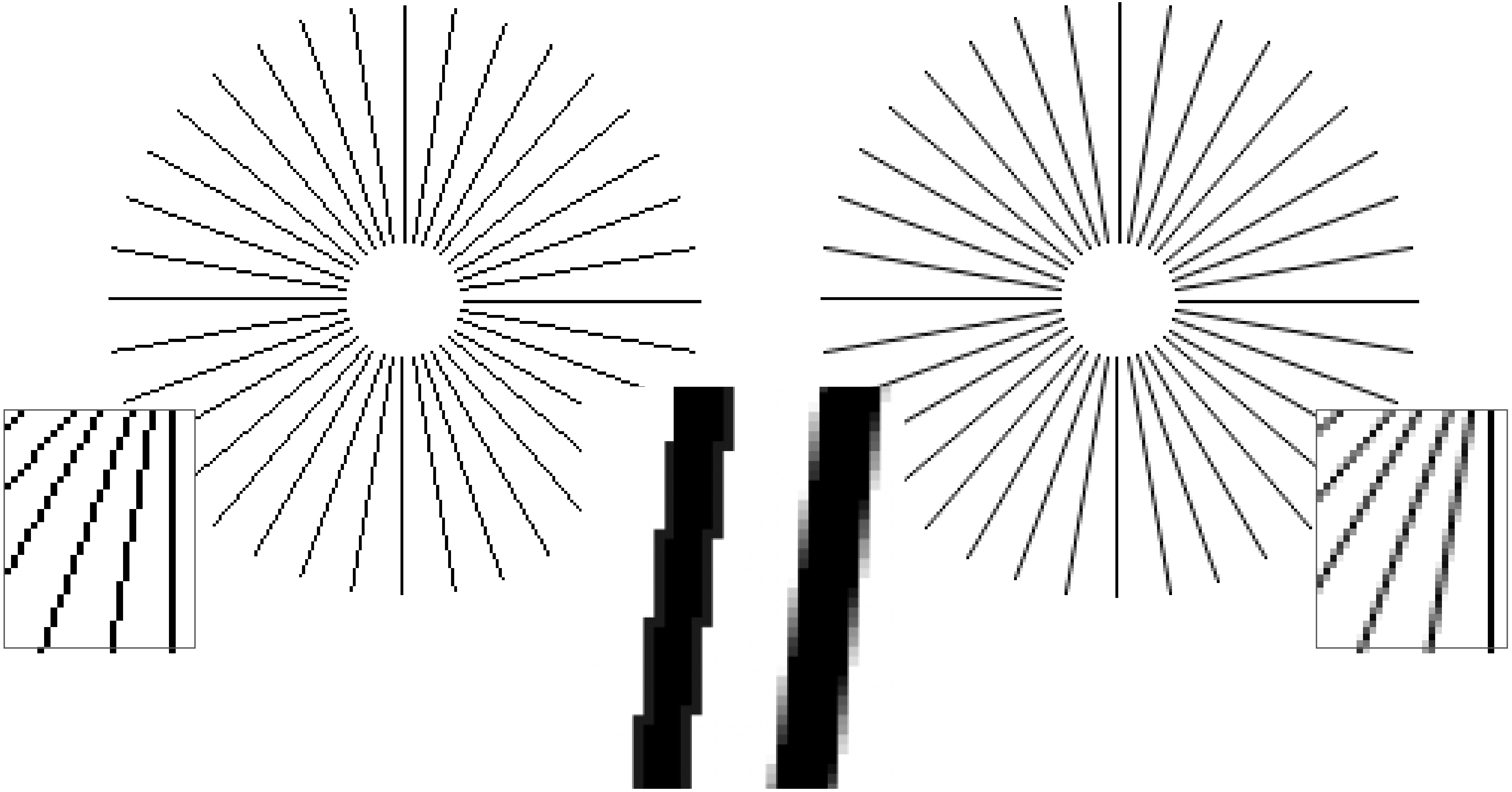
line of finite width

9 = max. intensity

...

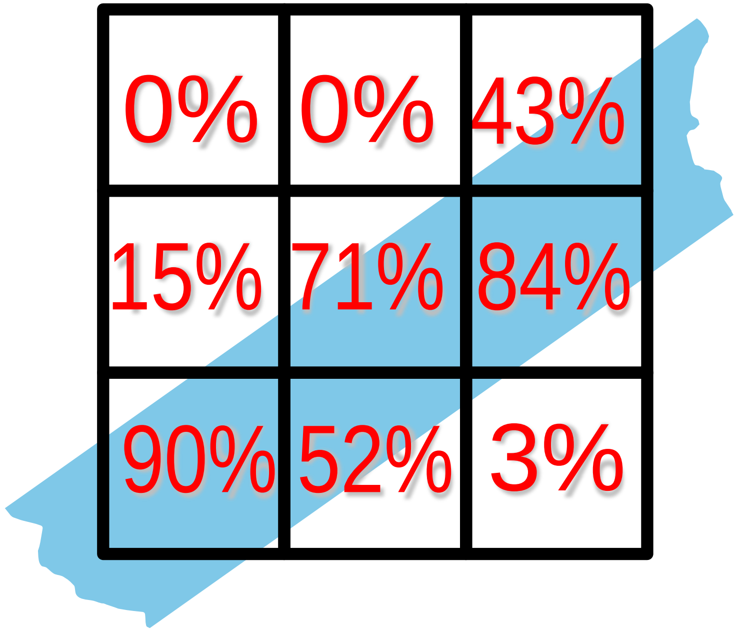
0 = min. intensity

Antialiasing



Antialiasing: Area Sampling Lines

- calculate the pixel coverage exactly
- can be done with incremental schemes



0%	0%	43%
15%	71%	84%
90%	52%	3%

Antialiasing: Pixel Weighting Masks

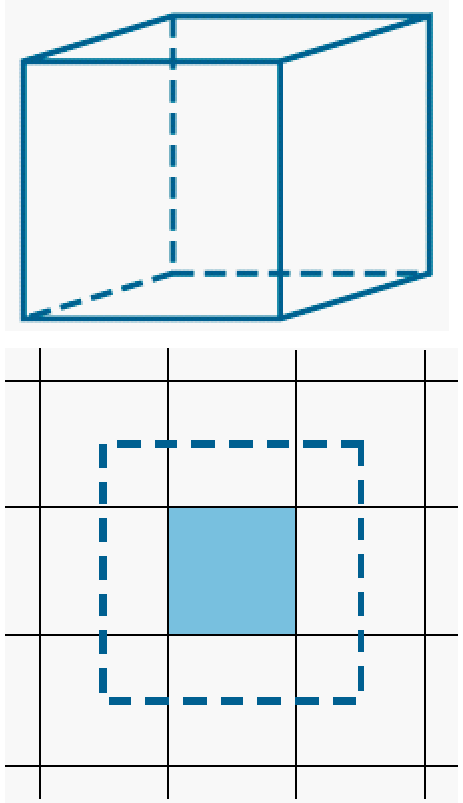
1	2	1
2	4	2
1	2	1

- more weight for center subpixels
- must be divided by sum of weights
- subpixel grids can also include some neighboring pixels

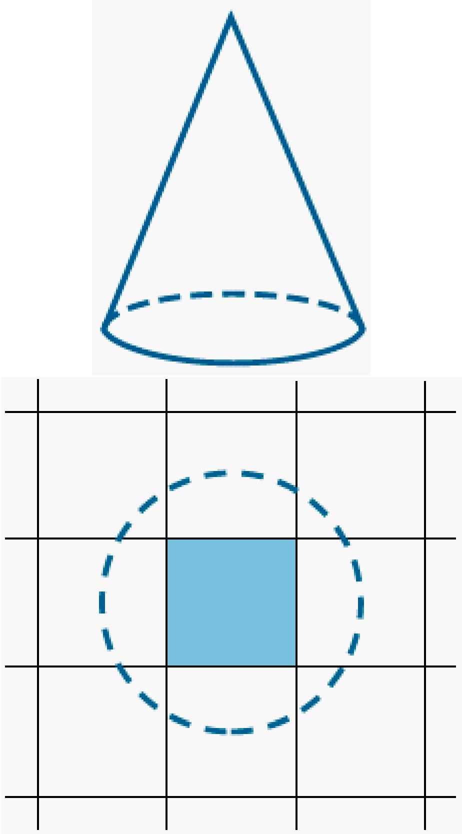
relative weights for a grid of 3x3 subpixels

Antialiasing: Filtering Techniques

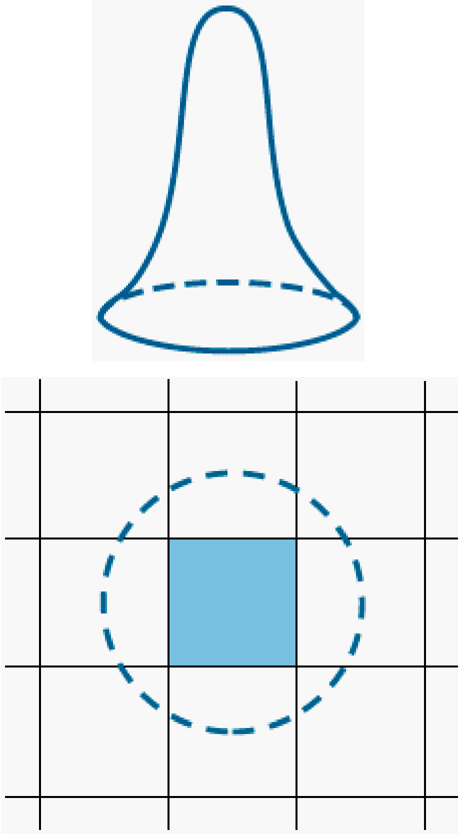
continuous overlapping weighting functions
to calculate the antialiased values with integrals



box filter

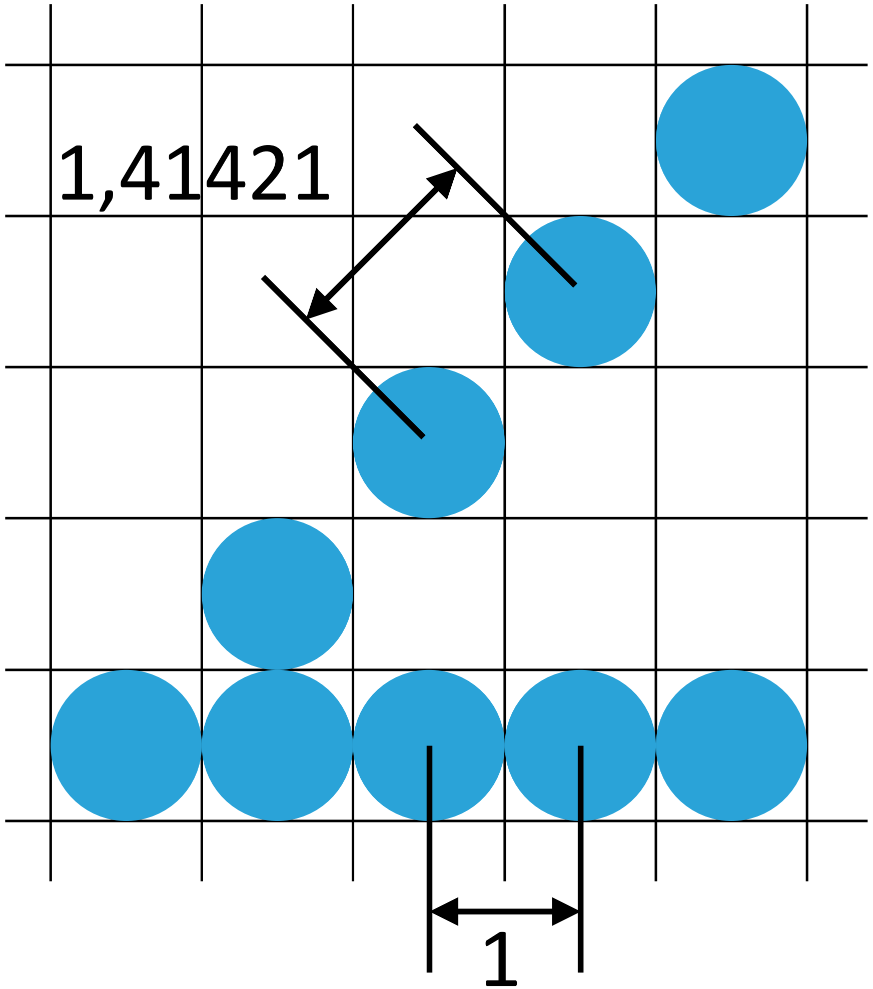


cone filter



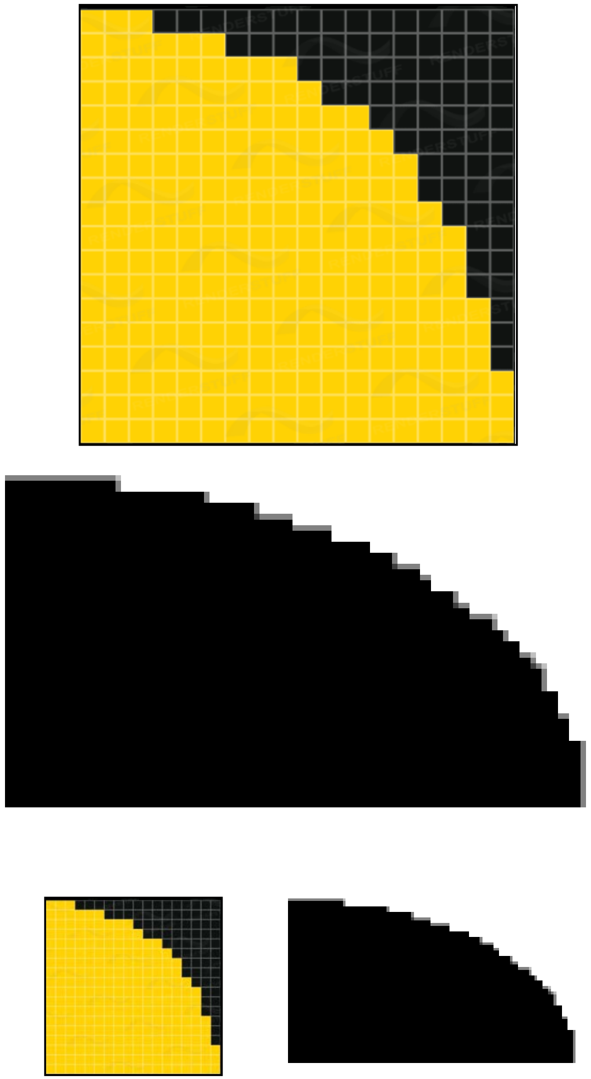
Gaussian filter

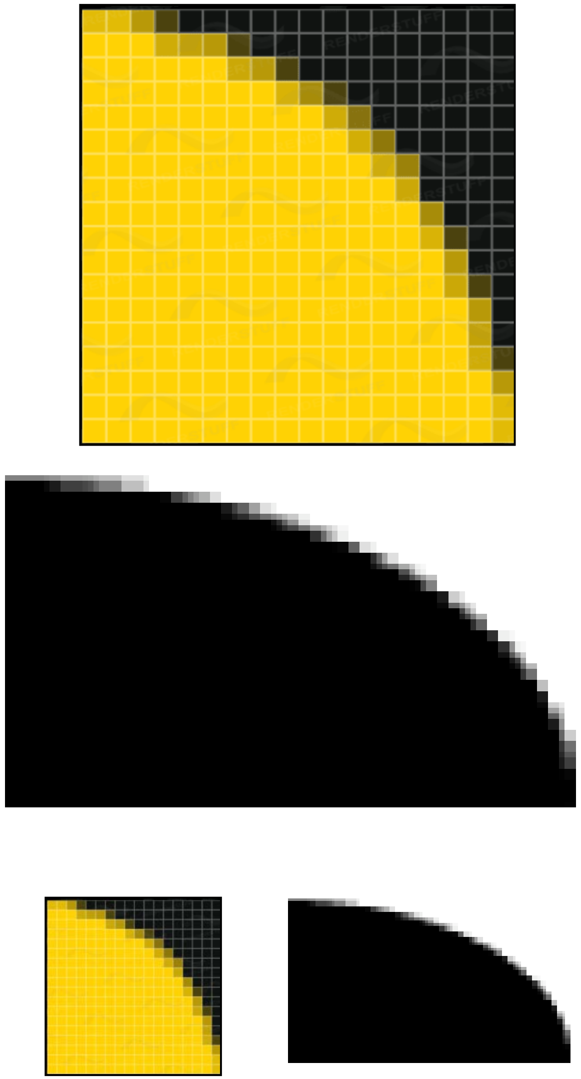
Antialiasing: Intensity Differences



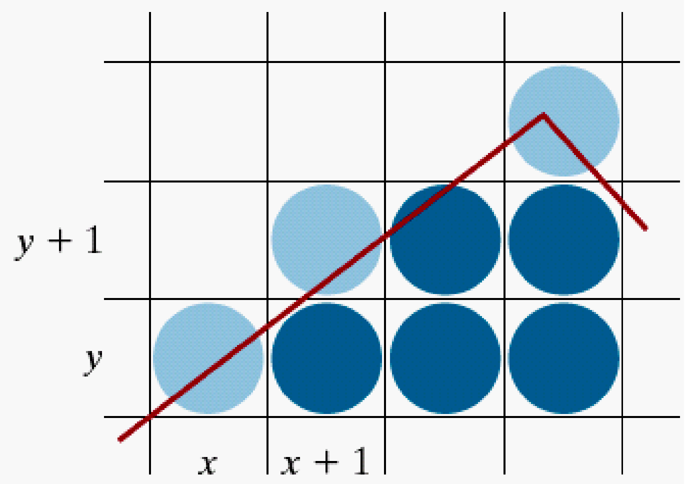
- unequal line lengths displayed with the same number of pixels in each line/row have different intensities
- proper antialiasing compensates for that!

Antialiasing Area Boundaries

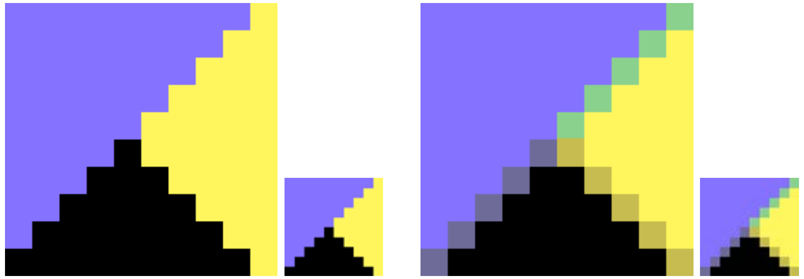




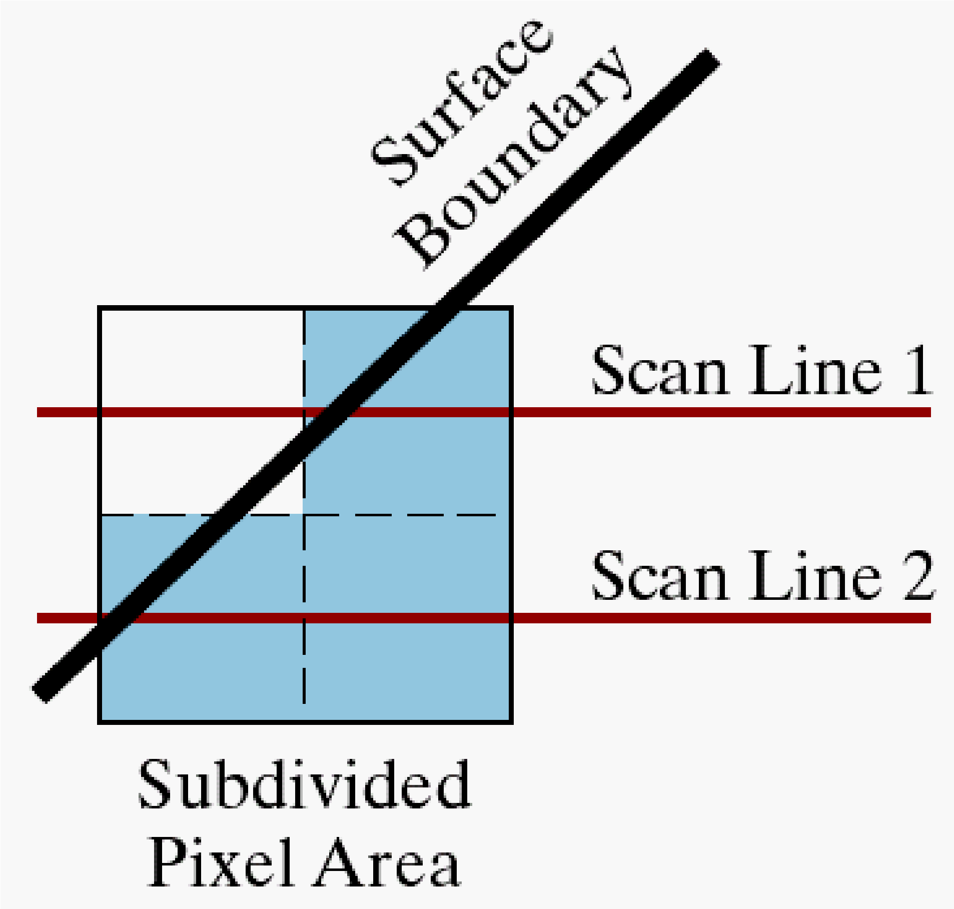
Antialiasing Area Boundaries (1)



adjusting pixel intensities along an area boundary



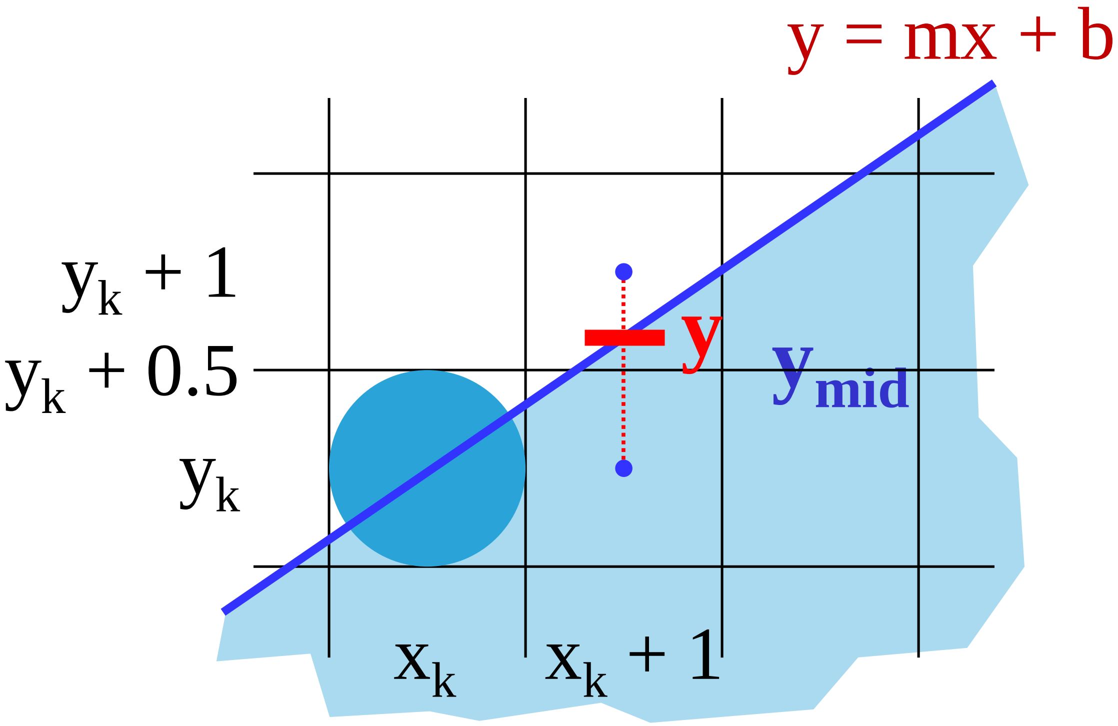
alternative 1:
supersampling



Antialiasing Area Boundaries (2)

alternative 2: similar to Bresenham algorithm

$$p' = y - y_{\mathbf{mid}} = [m(x_{\mathbf{k}} + 1) + b] - (y_{\mathbf{k}} + 0.5)$$



$p'<0 \Rightarrow$ y closer to $y_{\mathbf{k}}$

$p'>0 \Rightarrow$ y closer to $y_{\mathbf{k+1}}$

$p = p' + (1-m)$:

$p<1-m \Rightarrow$ y closer to $y_{\mathbf{k}}$

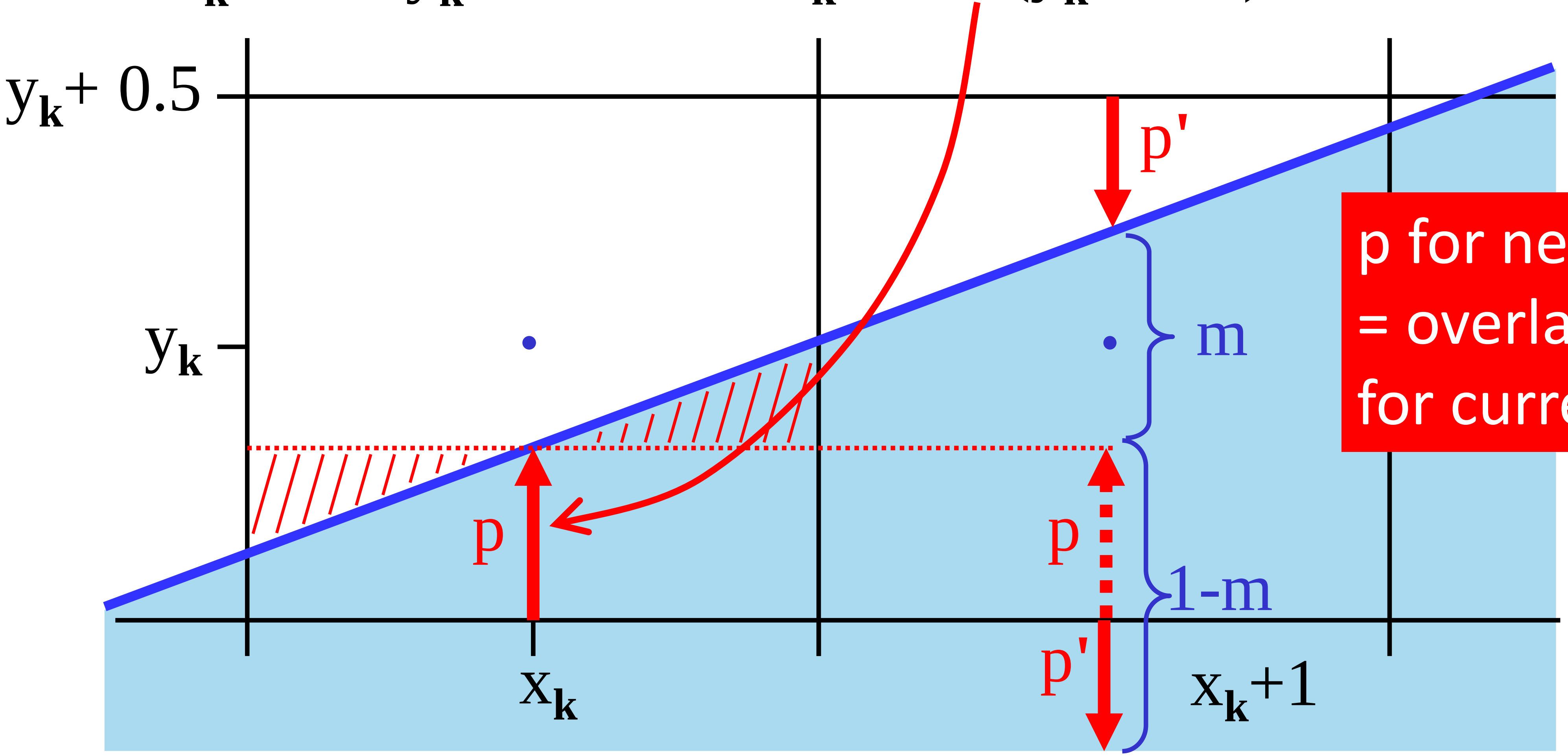
$p>1-m \Rightarrow$ y closer to $y_{\mathbf{k+1}}$

(and $p \in [0,1]$)

Antialiasing Area Boundaries (3)

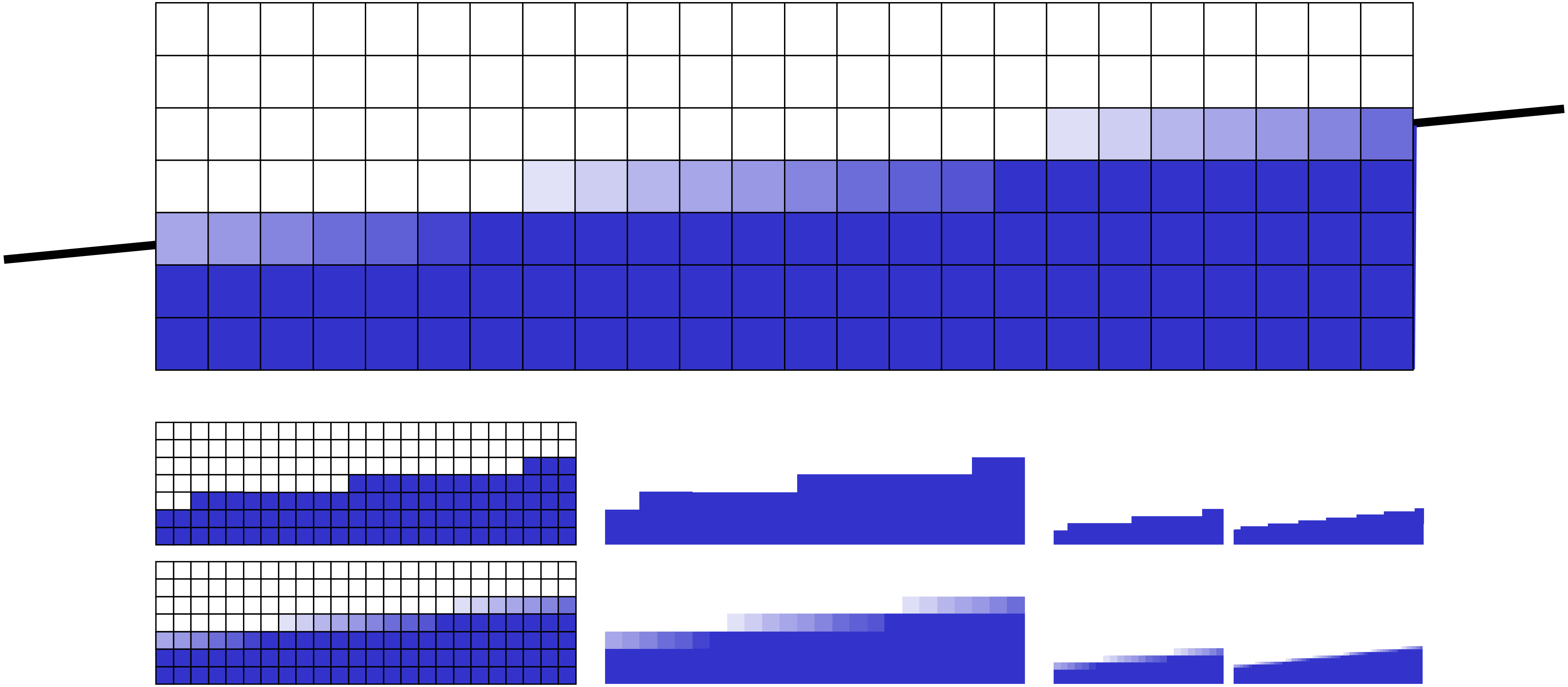
$$p = p' + (1-m) = [m(x_k + 1) + b] - (y_k + 0.5) + (1 - m) =$$

$$= mx_k + b - y_k + 0.5 \quad = mx_k + b - (y_k - 0.5)$$

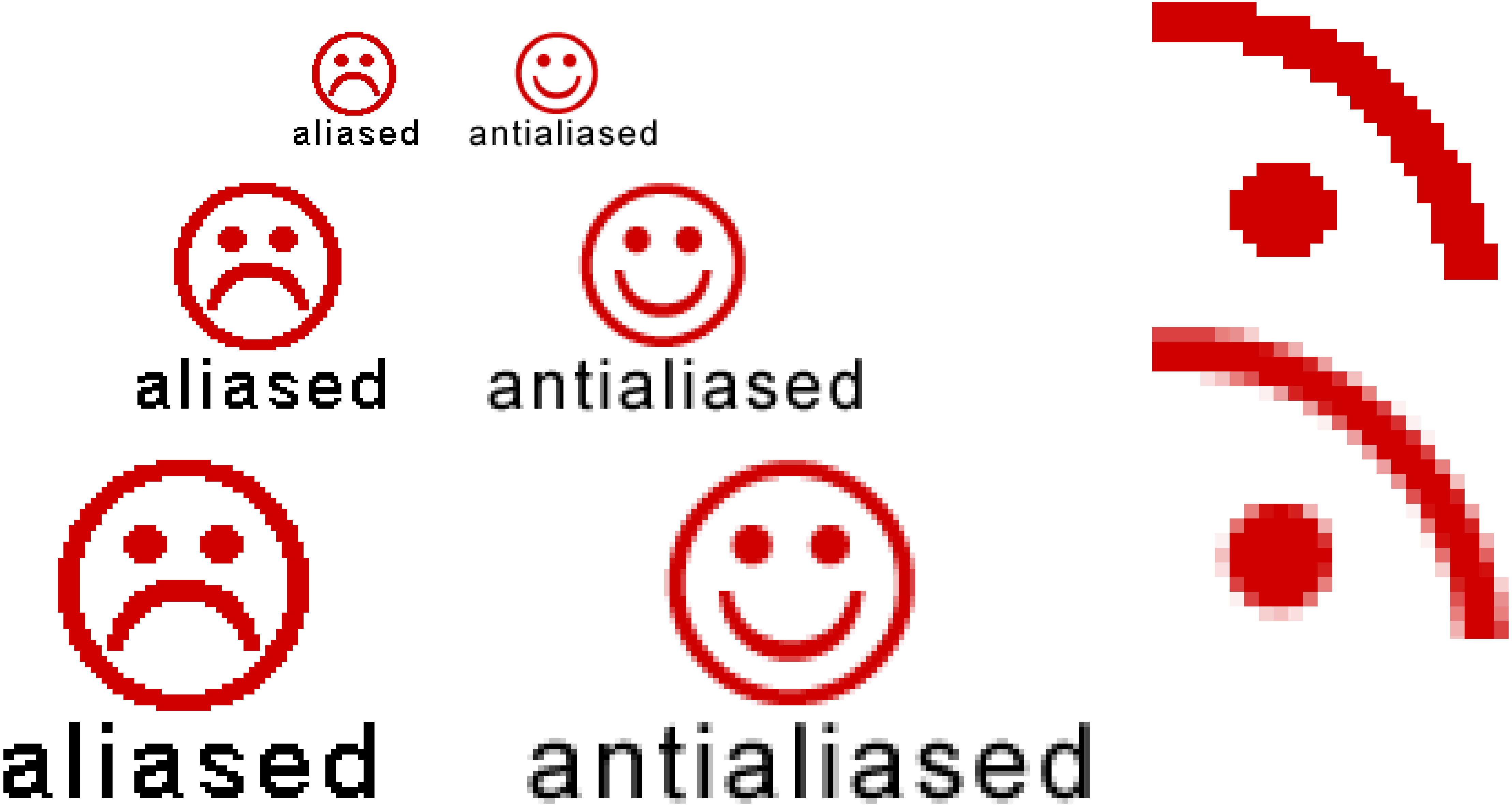


p for next pixel
= overlap area
for current pixel

Antialiasing Area Boundaries (4)



Antialiasing Examples



Werne